Anne Mosvold Ørke

Giving the Police a Head Start

Norwegian Named Entity Recognition Dataset and Model Development for Investigative Purposes

Master's thesis in Information Security (MIS)
Supervisor: Kyle Porter
Co-supervisor: Katrin Franke
August 2023

Anne Mosvold Ørke

Giving the Police a Head Start

Norwegian Named Entity Recognition Dataset and
Model Development for Investigative Purposes

Master's thesis in Information Security (MIS)
Supervisor: Kyle Porter
Co-supervisor: Katrin Franke
August 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Giving the Police a Head Start: Norwegian Named Entity Recognition Dataset and Model Development for Investigative Purposes

Anne Mosvold Ørke

Preface

First of all, I want to thank my supervisor Kyle Porter for all the help, support, guidance and understanding. I also want to thank my co-supervisor Katrin Franke for her support.

In addition I would like to thank Mads Skipanes for good input to the thesis and an interesting discussion about the topic.

Anne Mosvold Ørke

Abstract

Almost everyone owns digital devices, and digital evidence have become an important part of investigations. The large amount of digital devices and evidence needing to be examined and analyzed leads to the *digital forensics backlog*, which can cause delays and challenges in investigations. There exists Named Entity Recognition (NER) models for investigative purposes in other languages and there are Norwegian NER models, but there has not been done any research on a Norwegian NER model for investigative purposes. To help mitigate this problem for the Norwegian Police we propose a Norwegian NER model for investigative purposes.

We have collected and annotated domain specific Wikipedia articles and have created a dataset that can be used to train a NER model for investigative purposes in Norwegian. We have annotated the dataset with both general and investigative labels. During different experiments with fine-tuning a BERT based Norwegian Natural Language Processing (NLP) model, we found an optimized model which achieved good results. Our optimized model achieved a precision of 0.904, a recall of 0.908 and a F1-score of 0.906. The entity type with the highest score is *vehicle*, one of the domain specific entity types, with a F1-score of 0.973. 4 of the entity types achieved F1-score over 0.9 and 7 achieved F1-score over 0.79. When comparing the results of our model with other models, both models of investigative purposes in other languages and a Norwegian general model, we saw that our model performs as well as these models.

We were able to provide a proof of concept demonstrating that creating a Norwegian NER model for investigative purposes is possible. In a real world scenario, such a model should be trained on real investigation data. A NER model can not be a 100% relied on, but it still has the potential to be a great tool for the police and could help investigators save a lot of time.

Sammendrag

Nesten alle eier digitale enheter og digitale bevis har blitt en viktig del av etterforskninger. Den store mengden med digitale enheter og bevis som må undersøkes fører til et etterslep, som igjen kan føre til forsinkelser og utfordringer for etterforskerne. Det har blitt laget Navngitt Enhetsgjenkjenning-modeller (NER) til bruk i etterforskning for andre språk og det har blitt laget norske NER-modeller, men det er ikke gjort noe forskning på norske NER-modeller til bruk i etterforskning. For å prøve å løse dette problemet for det norske politiet foreslår vi en norsk NER-modell til bruk i etterforskning.

Vi har samlet inn og annotert Wikipedia artikler relatert til kriminalitet og har laget et datasett som kan brukes til å trene en NER-modell for norsk som kan brukes i etterforskninger. Vi har annotert datasettet med både generelle etiketter og etiketter spesifikke for etterforskning. Gjennom forskjellige eksperimenter med å finjustere en BERT-basert norsk NLP-modell (Naturlig Språkprosessering) fant vi en optimalisert modell som fikk gode resultater. Modellen vår fikk en presisjon på 0.904, en dekning på 0.908 og en F1-score på 0.906. Enhetstypen som gjorde det best var kjøretøy, som er en av de spesifikke etikettene for etterforskning, som fikk en F1-score på 0.973. 4 av enhetstypene fikk en F1-score på over 0.9 og 7 av dem oppnådde en F1-score over 0.79. Når vi sammenlignet vår modell med andre modeller, både modeller til bruk i etterforskninger for andre språk og en norsk generell modell, så vi at vår modell gjorde det like bra som disse modellene.

Vi klarte å lage et konseptbevis for at det er mulig å lage en norsk NER-modell til bruk i etterforskning. For å bruke modellen i faktiske etterforskninger burde modellen være trent på reelle etterforskningdata. Man kan ikke stole 100% på en NER-modell, men den har fortsatt potensialet til å bli et verdifullt verktøy for politiet og den kan hjelpe etterforskere med å spare mye tid.

Contents

Figures

Tables

Code Listings

Acronyms

AI	Artificial Intelligence. 2, 7, 11, 15, 47
BERT	Bidirectional Encoder Representations from Transformers. v, 9, 12, 13, 18, 30, 40–42, 49
BiLSTM	Bidirectional Long Short-Term Memory. 20
BIOLU	Beginning-Inside-Outside-Last-Unit. 9
CNN	Convolutional Neural Network. 20
CRF	Conditional Random Fields. 9, 17, 19, 20, 41
DL	Deep Learning. 7
ELMo	Embeddings from Language Model. 13
EU	Europeian Union. 47
IOB	Inside-Outside-Beginning. xi, 9, 10, 19, 23, 25
IOBE	Inside-Outside-Beginning-End. 9
IOBES	Inside-Outside-Beginning-End-Start. 9, 20
IOBS	Inside-Outside-Beginning-Start. 9, 23
LDA	Latent Dirichlet Allocation. 16
LSI	Latent Semantic Indexing. 16
LSTM	Long Short-Term Memory. 9, 10, 19
LSTM-CRF	Long Short-Term Memory and Conditional Random Field. 9, 19, 40
ML	Machine Learning. 2, 7, 9, 14, 15, 47
MLM	Masked Language Modeling. 12

NDT Norwegian Dependency Treebank. ix, 13, 19, 20

NER Named Entity Recognition. v, ix–xi, 2–5, 7–9, 11–21, 23, 25, 26, 30, 39–50

NLN National Library of Norway. 13

NLP Natural Language Processing. v, ix, 2, 7, 12, 13, 15, 16, 19, 30

NoTraM Norwegian Transformer Model. 13, 30

NSP Next Sentence Prediction. 12

POS Part-of-Speech Tagging. 8, 12, 13, 16, 30

ReLU Rectified Linear Unit. 10

SVM Support Vector Machine. 9, 16, 18

Chapter 1

Introduction

1.1 Background and Motivation

More and more of the world's population now owns digital devices, and a lot of people own more than one. This, alongside increased storage capacities, leads to a lot of digital evidence in investigations which needs to be examined to identify potential evidence. Digital evidence is gathered in all kinds of investigations and can, as all other kinds of evidence, lead to major breakthroughs in an investigation [1].

Investigators collect a lot of different types of digital evidence, like text, messages, images, and videos [1]. This can be gathered from a large variety of devices, for instance smartphones, laptops, social media and smartwatches. Lately the amount of collected digital evidence in investigations have increased, in addition digital devices and evidence that needs examining is collected in a lot more cases [2]. This results in a large amount of evidence that needs to be examined. Digital forensics, examining and analyzing all the digital evidence, can be very time consuming and take a long time. Digital Forensics Laboratories therefore have a hard time keeping up with all the digital evidence that needs examination. This leads to the *digital forensics backlog*, where devices and evidence are waiting to be examined and analyzed. The digital forensics backlog could further cause delays in investigations [3].

If investigations are slowed down it can potentially result in harmful consequences. Casey *et al.* [3] discusses that criminals might get time to commit more crimes or hide additional evidence before the investigators find them. In addition they discuss that it might be more difficult for the investigators to perform interviews with suspects if they do not have access to all the evidence. It might be harder to design interrogating questions and the suspect might easier confess if conclusive evidence is presented. Digital devices can also contain time-sensitive evidence which can help prevent additional harmful crime. Casey *et al.* [3] also discusses that if the investigators do not get access to this evidence in time, people might get hurt. Getting access to all evidence early in the investigation could help the investigators to locate additional important evidence before it gets deleted or lost. If someone is falsely accused it can cause a lot of harm to them if evidence is not examined in time and they are not acquitted. Their reputation might be harmed, they might lose their job and lose contact with friends and family [3].

In a perfect world the investigators should examine all the gathered evidence. But with the time limitations they have, going through all the evidence could lead to the examination being done in a rush, which again can lead to mistakes or compromised evidence. If the investigators on the other hand do not go through all the evidence it may lead to the most valuable evidence not being found. There is therefore a need for a method to help examining evidence more efficiently [3]. Both Scanlon [2] and Casey *et al.* [3] discuss methods that can be used to deal with the digital forensics backlog. But even with these methods there is a need for more ways to make examining and analyzing digital evidence more efficient.

The digital forensics backlog can be seen as a big data problem, and Artificial Intelligence (AI) can often be used to solve these kinds of problems. Du *et al.* [4] have looked at different AI approaches in digital forensics. Among other things, they look at file fragment classification and a data mining and ML approach to prioritize devices. They also look at using AI for encrypted evidence and evidence reconstruction. For classifying file fragments, one approach that is discussed is using Natural Language Processing (NLP). NLP is a subfield of AI that processes human language and includes both speech and text [5].

Named Entity Recognition (NER) is a subfield of NLP, which according to Mohit [6] is "the problem of locating and categorizing important nouns and proper nouns in a text". A NER model will identify and classify different kinds of nouns and proper nouns, like *names* and *locations*. This can be of great value in an investigation to locate interesting words and phrases in text without explicit need of a keyword list, or prior knowledge of specifics of the case. Investigators often encounter large amounts of texts which are infeasible to examine thoroughly. The current approach to finding relevant information in text without reading through everything is keyword and regex searches. But in many cases the investigators may not have the necessary knowledge and information that is needed to know what they should search for [7]. NER is a ML approach to solve this problem, which can identify possibly valuable and relevant information without having to explicitly search for it.

Chau *et al.* [8] and Wu *et al.* [9] have looked at using NER for investigative purposes and they have achieved promising results. Jørgensen *et al.* [10] have published a dataset and created a model for Norwegian NER, but as of now there are no Norwegian NER models for investigative purposes. Using NER could be very valuable for the Norwegian Police and it could help make processing text in investigations more efficient. Since it does not exist any model for this specific domain in Norwegian, the police will have to create their own annotated dataset and train their own model if they are interested in using NER in their investigations. Since such a model would need to be trained on real forensic data, which can not be published publicly due to privacy concerns, the police need to create the model themselves. But it would still be of great value for the police if there existed a proof of concept which could show them if it is possible to create an efficient model and how that could be done. We are therefore doing research in a similar domain, but will less sensitive data, which still has the potential to be applicable to the police and serve as a proof of concept.

Using NER in investigations could be very useful for investigators and help in different ways. Chau *et al.* [8] looks at using NER to find valuable entities in police narrative reports, which can be used for further analyses in an investigation or for crime pattern recognition to avoid future incidents. Wu *et al.* [9] created a NER model for use in public opinion monitoring to help law enforcement alert about crimes before they happen. When it is not feasible to read through all the evidence, NER can help to identify valuable and important evidence in large amounts of text. It can extract words from the texts, which then also can be a possible means of triage. The extracted words can give an indication to which texts could contain valuable evidence.

Being able to use NER to prioritize the evidence and be more efficient in finding conclusive evidence could be very valuable and useful for law enforcement and digital forensics laboratories and it could help them to reduce the digital forensics backlog. Getting rid of the digital forensics backlog and speeding up investigations could help avoid the harmful consequences of a delayed investigations and is of great benefit for both the law enforcement, the investigators and the general public.

A NER model can be used on both collected evidence, on reports and transcribed interviews. By identifying the named entities in the texts it can help the investigators to find important parts faster. Instead of reading through the whole text to see if it mentions something important they can look at the identified named entities. It could be very valuable if the investigators have collected a large amount of text messages to be able to automatically check if there is any mentions of for instance drugs or weapons. If they are for instance looking for a car in relation to a crime they are trying to solve, it would be very valuable to be able to extract any mentions of specific cars from the text they have available without having to read through everything. A NER model could be a great tool for investigators to locate where interesting information can be found in the evidence and texts they have, and hence help them to prioritize the evidence. A NER model could also be used for tasks such as crime detection and prevention such as in Wu *et al.* [9].

1.2 Research Questions

Using a NER model in investigations could be very interesting for the Norwegian Police. There are already NER models for investigative purposes and Norwegian NER models, but there have not yet been created a Norwegian NER model for investigative purposes. The purpose of this research project is to create such a model. We also want to look at what performance we can achieve. We therefore have the research questions:

1. How can we develop a Norwegian NER model fine-tuned for investigation, with the restriction of using open-source data?
2. What kind of precision, recall and F1-score could such a model acheive and how does it compare with the state-of-the-art Norwegian NER model trained for general purposes and NER models created for investigative purposes in other languages?

1.3 Contributions

The contributions of this master thesis is a dataset and fine-tuned Norwegian NER model for investigative purpose. We have also done experiment on what kind of performance the model can achieve. Before we created the dataset we first needed to define the labels we were going to use, which means what categories of words the NER model is going to classify. We talked to an investigator and consulted literature to figure out what labels were most reasonable to use. We then created a dataset with crime related text collected from Wikipedia and added the general and investigative NER labels, before we fine-tuned a pretrained model with the domain specific dataset we created. We have quantitatively evaluated the model and measured the precision, recall and F1-score and were able to create a model with good performance. Our model achieved scores as well as other NER models for investigative purposes and the Norwegian NER models. The best F1-score we achieved for the different entity types were 0.973 for *vehicle* and 0.949 for *person*. To other entity types also got a F1-score over 0.9 and 4 more got a F1-score over 0.79. The overall performance our model achieved was a precision of 0.904, a recall of 0.908 and a F1-score of 0.906. The equivalent Norwegian NER model, with just general entity types, got a F1-score of 0.91. This is a little bit higher than our score, but for the different entity types our model performed better on some of them. When it comes to other domain specific NER models our model performs better than several of them.

1.4 Thesis structure

The thesis is structured in the following chapters:

Introduction The introduction chapter gives an introduction of the problem and topic. The research questions and the contribution are also introduced.

Background This chapter consists of the background theory that is needed to understand the research that has been done, like NER and Transformers.

Related Work Related work reviews the research that has already been done on the area, including NER for investigative purposes in other languages and Norwegian NER models.

Methodology The methodology chapters presents the methods we used and how we created the dataset and trained the model. It reviews all the parts of the process and also how we evaluated the model.

Results This chapter presents the results from the experiments we did with the dataset and model we created.

Discussion The discussion chapter discusses the results and compares it to other similar research. It also looks at how a Norwegian NER model for investigative purposes could be used and what limitations it has. This chapter also suggests future work and answers the research questions.

Conclusion The last chapter concludes the research.

Appendix This chapter contains additional documents, like the full scripts that have been used.

Chapter 2

Background

This chapter will go over the background theory that is needed to understand this master thesis. It will first cover some general concepts like Neural Networks and Natural Language Processing (NLP), before it goes into Named Entity Recognition (NER), Transformers and language models. It also covers evaluation metrics.

2.1 Neural Networks

Neural networks is a very popular method for ML. Neural networks can learn from input data and can then be used to perform tasks, both for classification and regression. A neural network consists of nodes, also referred to as a neuron, and links. The nodes are connected by directed links and each link has a weight. The neural networks are arranged in layers and nodes in one layer receives input from nodes in the previous layer. The layers between the input and output layer are called hidden layers, and a neural network can have multiple hidden layers [11]. Most standard neural networks only have 2 or 3 hidden layers, but some have up to 150 hidden layers and they are called deep neural networks. Deep neural networks are considered Deep Learning (DL) [12]. Each node in the network will compute the weighted sum of the input to the node and then use an activation function to compute the output of the node. The output layer will compute the final output of the neural network. To train a neural network, the error in the output is used to back-propagate through the network and compute new weights. When sending input into a neural network, the neural network will give an output with the predicted value for regression or the probability of the instance belonging to the different classes for classification [11]. Figure 2.1 shows a simple illustration of a neural network.

2.2 NLP

NLP is the part of AI that regards processing natural human language. It regards both text and speech and uses linguistics, statistics, ML and DL to create

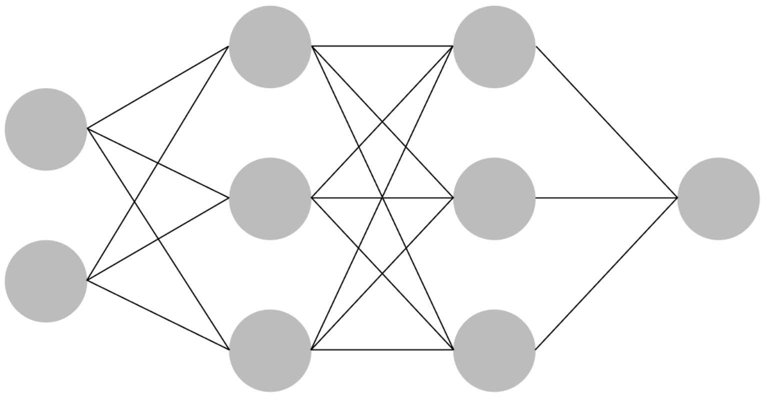


Figure 2.1: Simple illustration of a neural network

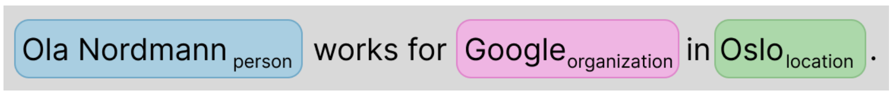


Figure 2.2: Example of NER tags on a sentence

algorithms and models that can perform a variety of tasks, like machine translation, Part-of-Speech Tagging (POS) and NER [5][13]. According to Ukwen and Karabatak [5] NLP "allows us to understand the structure and meaning of natural human language by analyzing different aspects of grammar, semantics, pragmatics, and morphology".

2.2.1 NER

Named Entity Recognition (NER) is a subfield of NLP that regards identifying different entities in a text. The entities can for instance be *person*, *location* and *organization*. Given a set of labels or categories, the goal of a NER model is to categorize each word in a text with the correct label/category [6]. Figure 2.2 shows an example of NER applied to a sentence.

There are different approaches to NER. The most basic approach is lexical lookup, where lexicons are created with the most popular entities. Phrases are then compared with the entities in the lexicon. The lexical lookup approach will find every occurrence of a phrase in the lexicon, but will not be able to recognize new entities. Other methods are therefore needed. A lot of systems uses more than one approach [8]. For some time, the most popular approach was rule-based, which can work well for small domains. The rule-based approach is based on some rules and a lexicon which determines if a text phrase is a named entity or not. The rules tells the system how to identify named entities, like that a capitalized first name and a capitalized last name is a *person*. This approach works well with few entity types, but does not scale to many entity types and is not as good with unstructured text.

When more and more data became available, a statistical approach became more popular. The statistical approach is based on annotated training data, where the training data is labeled with the corresponding entity type. A statistical model is then used to find the probability of the text phrase being specific entity types [6][8].

The last approach is Machine Learning (ML), where there is no need to create any lexicon or rules. This has become the most used technique when it comes to NER. Instead of creating rules manually, which still might not be able to detect all named entities, ML algorithms will use training examples to learn how to detect all named entities in the best way possible. There exists a lot of different machine learning algorithms, and a lot of them can be used for NER [8][14]. This includes, among others, decision trees, Support Vector Machine (SVM) and neural networks [14]. Newer methods that can be used for NER is Long Short-Term Memory (LSTM), Conditional Random Fields (CRF) and the combined method Long Short-Term Memory and Conditional Random Field (LSTM-CRF), which are also neural architectures. Recently Transformers and Bidirectional Encoder Representations from Transformers (BERT) have become very popular. All these methods need training data [15].

The preferred algorithm to use for NER can be different based on what the goal is. Some algorithms are often more popular than others, and this changes over time [8] Today Transformers and BERT are some of the most popular approaches to use for NER [16][17].

Annotation styles

When creating datasets for training NER models, a way to identify which words or phrases are a named entity is needed. There exists multiple annotation styles for this. The annotations are added to each word, or token, in the dataset to indicate if it is a part of a named entity or not. The most basic annotation style is Inside-Outside-Beginning (IOB), but there exists many other variations. With the IOB annotation style a B-label is added to the token if it is the start of a named entity and the previous token have the same entity type but is not a part of the same entity. The I-label is used on tokens that are a part of a named entity and is not tagged with a B-label. The O-label identifies tokens that is not part of a named entity [18]. How the O-label is used can be seen in Figure 2.3.

The Inside-Outside-Beginning-End-Start (IOBES) annotation style have the same I, O and B labels, but in addition they have the E and the S labels. The E-label is the end-token of an entity with multiple words, while the S-label is used on entities with only one token. This style is sometimes also called Beginning-Inside-Outside-Last-Unit (BIOLU), where the L is the last token in a entity and U stands for unit length entities. There also exists IOB variations with just the E-label, Inside-Outside-Beginning-End (IOBE), and just the S-label, Inside-Outside-Beginning-Start (IOBS) [10].

Some NER models is trained with the IOB2 style, which is very similar to the

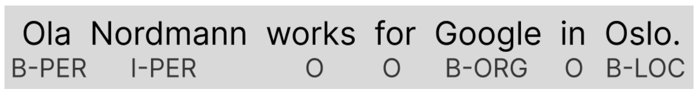

Ola Nordmann works for Google in Oslo.
B-PER I-PER O O B-ORG O B-LOC

Figure 2.3: Example of a sentence annotated with the IOB2 style

IOB style. The only difference is that for all named entities, the first token in the named entity is tagged with the B-label [10]. An example of a sentence annotated with the IOB2 style is provided in Figure 2.3.

2.2.2 Transformers

For a long time recurrent neural networks and LSTM were the standard approaches for processing language [19]. Vaswani *et al.* [19] proposed a new architecture, the Transformer, which is more computational efficient and have improved performance. The Transformer architecture uses an encoder-decoder structure, like a lot of other models. In a encoder-decoder structure, the encoder encodes the input, assigns numbers to the words, before the decoder decodes the input into an output value, transform the numbers back to words. Both the encoder and the decoder uses fully connected self-attention layers, where all nodes in one layer is connected to all nodes in the next layer. Attention is a mechanism that allows the model to learn which input is most important.

The encoder in the Transformer architecture has 6 layers. Each of the layers consists of a self-attention mechanism and a feed-forward network. The decoder also has 6 layers, with a self-attention mechanism, a feed-forward network and mechanism that does multi-head attention on the output from the encoder. The attention functions computes an output based on the weighted sum of values in a set of key-value pairs. The weights are based on a compatibility function, a function that computes the probability different outputs, of a query and a corresponding key. In the encoder the self-attention layer uses the output from the previous layer in the attention function. The self-attention layer in the decoder is a bit different from the one in the encoder. It is modified to make sure that the predictions only rely on the output from the positions up to the current position in the sequence of text. The feed-forward networks in the encoder and the decoder is fully connected. Two linear transformations and a Rectified Linear Unit (ReLU) activation is applied to all the positions in the sequence [19]. An illustration of the Transformers architecture is provided in Figure 2.4.

The advantages of the self-attention layers in this architecture is that the computational complexity is lower than with other methods. In addition a lot more computation can be parallelized. The translation models Vaswani *et al.* [19] created and tested performed a lot better then previous models. They also tested if the Transformer architecture could be used for other tasks than translation and they achieved good performance on that as well [19].

After the creation of the new Transformer architecture, this has been a very popular approach for NLP and is widely used for all NLP tasks. It has a lot of

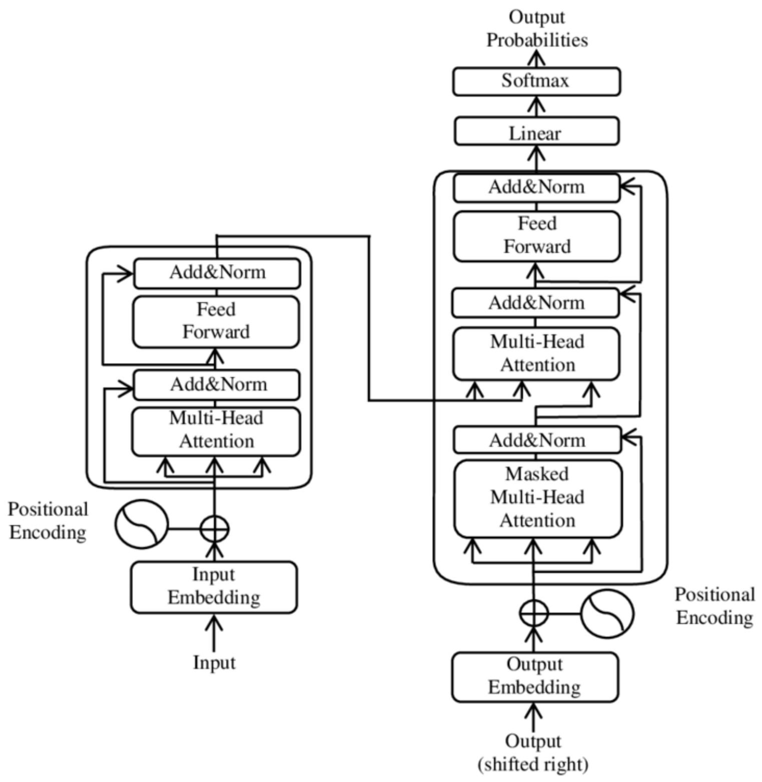


Figure 2.4: An illustration of the Transformers architecture, which shows the encoder and the decoder with the attention functions. Yuening Jia[20], CC BY-SA 3.0 https://creativecommons.org/licenses/by-sa/3.0, via Wikimedia Commons

advantages, like efficient training and it is scaling well, and in addition it works well with long sequences. The Transformer architecture is also beneficial for pretraining. Pretraining means that models are trained on a general dataset which can then be easily further trained, fine-tuned, to specific tasks with good performance. A pretrained model can be fine-tuned to a lot of different tasks, like text classification, translation and NER [16].

Wolf *et al.* [16] created a library in Huggingface, an AI community, called *Transformers*, where the Transformer architecture is implemented and pretrained models are available.

2.2.3 Language Models

According to Kapronczay [21] a language model is "a probability distribution over words or word sequences". These probabilities say something about how likely it is that this word fits into the sentence based on the data the language model is trained on. Language models are used to process language and gives an understanding of the natural way language is used that makes machines able to do a lot of different tasks in NLP [21]. These tasks includes for instance translation, POS, answering questions and NER.

In traditional language models, which are unidirectional, the probability of a word is dependent on the previous words in the sequence. This is not always a good solution, since the context of a word is often understood by the next words in the sequence. In addition this approach is not as well suited for fine-tuning and it is not very scalable [21][17].

New bidirectional language models have been proposed that take both the previous and next words in the sequence into consideration. These are masked language models. Masked language models will mask some of the words in the input randomly, before it predicts those words. It is trained to predict the masked words by taking the context of the words before and after into account. These kinds of models is fitting for tasks where the context is important, like NER and translation, since it looks at the words in both directions. The traditional unidirectional language models on the other hand is more fitting for tasks where text is generated, since it there is no words after the current to take into consideration [17][22].

BERT

Devlin *et al.* [17] discusses that the unidirectional language models have limitations when it comes to pretrained models. As previously mentioned it is sometimes important to take the context in both directions in the sequence into consideration. They therefore present a new language representation model called Bidirectional Encoder Representations from Transformers (BERT), which is a masked and bidirectional language model. BERT is pretrained on unlabeled data. All the parameters in the pretrained model can then be fine-tuned with labeled data to be fitted to different tasks. The architecture behind BERT is based on the Transformer architecture Vaswani *et al.* [19] proposed. According to Devlin *et al.* [17]"BERT's model architecture is a multi-layer bidirectional Transformer encoder". There are mainly two BERT models, $\mathbf{BERT_{BASE}}$ and $\mathbf{BERT_{LARGE}}$. The $\mathbf{BERT_{BASE}}$ model has 12 layers, while the $\mathbf{BERT_{LARGE}}$ model has 24 layers.

The input to BERT models may be either just one sentence or two sentences, making it able to perform a large variety of tasks [17].

When Devlin *et al.* [17] pretrained the BERT model, they pretrained it on two unsupervised tasks, Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The MLM part is the reason that BERT is bidirectional and is done by randomly masking 15% of the input. The model will then predict the masked input. Many NLP tasks deals with the relationship between sentences. By just using MLM the relationship between sentences is not captured. Therefore it is also pretrained on NSP. Two sentences is then used as input, together with a label that says if the second sentence follows the first sentence or not. For 50% of the dataset the sentences follow each other. BERT is trained on BookCorpus [23] and Wikipedia articles [17].

Because of the Transformer architecture, BERT can easily be fine-tuned to a lot of different tasks, and because of the way it is pretrained it can be fine-tuned to tasks regarding both single sentences and pairs of sentences. In the process of fine-tuning, all the parameters are modified to fit the specific task by sending in inputs and outputs for the specific task into the model. Since the BERT model is already pretrained, fine-tuning it for specific tasks do not require a lot of computational resources [17].

Devlin *et al.* [17] tested BERT on 11 different tasks. They fine-tuned BERT for different tasks and tested the performance of the fine-tuned model. In all cases the BERT model performs better than previously tested models.

Norwegian Language Models

Kummervold *et al.* [24], with the project Norwegian Transformer Model (NoTraM), created BERT-based pretrained language models in Norwegian. They used texts from the National Library of Norway (NLN) to create a large Norwegian corpus. The corpus they created consisted of both the written variations of Norwgian, 83% Bokmål and 12% Nynorsk, and has 109GB of Norwegian text. As Devlin *et al.* [17] did, Kummervold *et al.* [24] also used masked language modeling to train their model. They wanted to create a model that could be fined-tuned to all kinds of tasks for Norwegian. The model they created, called *nb-bert-base*,[1] outperformed existing models [24].

Kutuzov *et al.* [25] have also created a Norwegian language model based on BERT, called NorBERT. They used news texts and Wikipedia texts, in total around 2 billion tokens and the model they created were a joint model for Bokmål and Nynorsk. They also created two models based on Embeddings from Language Model (ELMo), which uses embedding and vectors, which had sligthly lower performance, while the computational time was a lot lower in terms of training. The NorBERT model was tested on different NLP tasks and compared to among others the nb-bert-base model. There is not very much difference in the scores and different models performs better for different tasks. While NorBERT performs best on sentiment analysis, nb-bert-base performs best on POS and NER [25].

2.3 Norwegian Dependency Treebank

For some NLP tasks, like POS, the syntactic and semantic of a sentence is needed. A treebank is a corpus that has annotated syntactic or semantic information. Solberg *et al.* [26] created a treebank for Norwegian called Norwegian Dependency Treebank (NDT). The NDT includes both Bokmål and Nynorsk, and it has 311 000 tokens in Bokmål and 303 000 tokens in Nynorsk. The corpus consists of different types of text, like government reports and blogs, but it mostly consists of newspaper text. The treebank have been annotated with morphological and syntactic information [26].

[1]https://huggingface.co/NbAiLab/nb-bert-base

2.4 Evaluation metrics

Precision, recall and F1-score are the standard evaluation metrics to use for classification tasks, especially when the data is imbalanced between positive and negative samples, and a lot of the research that is discussed in Section 3 uses these metrics. Precision tells how precise a model is. In a binary classification problem with the classes positive and negative, precision calculates how many of the predicted positives are actually positive. In a multiclass classification problem it calculates how many of the predicted instances of a class is classified correctly. Precision is very useful in situations where classifying an instance as the wrong class is harmful [27].

Recall regards if the model is able to capture all the instances of a class. It calculates how many instances of an entity the model classified correctly. This means how many of the actual positives the model classifies as positive. This is valuable when it is important to detect every instance of an entity [27].

Accuracy is often used as a evaluation metric, but in situations where there are large differences between the number of instances in each class, like it is in NER, the F1-score is a more fitting metric. The F1-score is a combination of the precision and recall and the formula is shown in Figure 2.5 [27]. These metrics gives a good overview of the performance of a ML classification model and gives a good basis for discussing how suited the model is for the problem at hand.

$$\frac{2 * precision * recall}{precision + recall}$$

Figure 2.5: Formula for F1-score

Chapter 3

Related Work

The related work chapter goes over what research is alredy done in the field. We first look at some more general research on using AI and NLP in investigations, before we look at previous research on using NER for investigative purposes. Lastly we look at research done on Norwegian NER.

3.1 Artificial Intelligence and Machine Learning in Investigations

AI can be very useful for law enforcement and can help mitigate the digital forensics backlog. There are a lot of different use cases for AI and ML in investigations and various methods can be used to make investigation more efficient. Garfinkel *et al.* [28] proposed a solution for using classification to determine the ownership of data when examining storage media. Marturana *et al.* [29] researched a method to prioritize mobile devices in an investigation and figure out if they contained evidence that a specific crime had been committed. They tried out different classification algorithms and different scenarios. The classification algorithms they did experiments on were Bayesian Networks, Decision Tree and Locally Weighted Learning. They then tested them with different quantities of training data and different test approaches, like k-folds cross-validation and a separate test set. The different scenarios achieved varying results, where one algorithm performed better in one scenario while another performed better in another, but the results still showed promise in the approach. Encrypted evidence can be a substantial challenge in investigations. AI can be used to retrieving the cryptographic key and learning valuable information about the data without decrypting it [4]. Khan *et al.* [30] researched a method for reconstructing the timeline of an event based on activities on a file system. This is done with artificial neural networks. Images can be very valuable in investigations and it has been done research on using AI to both identify objects in images and detect forged images [4]. All these methods could help investigators save a lot of time.

3.1.1 Natural Language Processing

NLP can also be very useful in investigations and there have been done some research on this. Fitzgerald *et al.* [31] used NLP techniques for file fragment classification using Support Vector Machine (SVM). File fragment classification can be very useful for carving, or reassembling, files that are not stored on the same place on a hard drive.[1]

Multiple researchers have looked at using topic modeling techniques in investigations. Du *et al.* [32] created a model based on Latent Semantic Indexing (LSI) and WordNet to find emails with interesting content. WordNet is a lexical database, while LSI is an indexing model which finds topics in documents. Topic modeling is something that can be used in investigations to find relevant documents in the evidence by identifying the topics in documents. Waal *et al.* [33] used a Latent Dirichlet Allocation (LDA) model to do this and concluded that topic modeling can be very useful for identifying the context of documents.

When it comes to mobile forensics, O'Day and Calix [34] created, and made publicly available, a text message corpus. The idea is that it can be used to develop NLP techniques that can be used to analyze messages on phones and on social media in an investigation. They also tested the corpus with a Naïve Bayes classifier to detect messages that were drug related [34].

A lot of interesting and relevant information can be found on social media and this information can be very valuable in an investigation. Keretna *et al.* [35] proposed a method for extracting features about writing style, with a POS tagger, to authenticate accounts on social media. Lau *et al.* [36] also looked at using NLP methods to analyze social media data. They created a supervised network mining method to detect cybercriminal networks on social media. Another research that has seen the importance of social media and online communication is the work of Sun *et al.* [37]. They created a digital investigation platform that used both unsupervised and supervised NLP methods to analyze messages and communication, and assist investigations. Their solution first takes in data from a communication network and represents it as a graph with individuals as vertices and messages as edges. Topic modeling is done on the messages before a classification algorithm is used to determine if something criminal is happening. The platform they created performs better than existing solutions with a F1-score of 0.65. Existing solutions only has a F1-score of 0.59. The precision of this system is 0.83, while existing solutions only achieves 0.58.

3.1.2 Named Entity Recognition

Earlier research has studied the application of NER for investigative purposes. Chau *et al.* [8] made a named entity extractor based on neural networks. They saw a need for more efficient methods to extract important and valuable information

[1]File carving is the process of reassembling computer files from fragments in the absence of filesystem metadata.

from police narrative reports and that the method could be used for crime pattern recognition as well. The system they created consists of three parts:

Noun Phraser The first part is a noun phraser which extracts the noun phrases that are later candidates for named entities.

Finite State Machine The next part is a finite state machine. The finite state machine does lexical lookup on the found noun phrases, and the word before and after. Then the phrases gets a score based on if a match is found.

Neural Network The last part is a neural network which then predicts the entity type of the noun phrases.

To figure out which entity types would be most useful in an investigation Chau *et al.* [8] talked to detectives and analysts at the Tucson Police Department. The entities they then concluded on using were *person*, *address*, *vehicle*, *narcotic drug* and *personal property*. The lexicons were also created mainly with data from the Tucson Police Department.

To test the system Chau *et al.* [8] did experiments on police narrative reports from the Phoenix Police Department. Since the data were real data it was quite noisy and had a lot of typos, which potentially make the extraction more difficult and lead to lower performance, but it also allowed them to test the system on actual data and it would give a more realistic performance measure. The results they got shows that the system is quite good at extraction *people* and *drugs*. For the *person* entity they got a precision of 0.741 and a recall of 0.734. When it comes to *drugs* they got a precision of 0.854 and a recall of 0.779. The *vehicle* entity type was not tested since there was not enough occurrences of it in the test data. The *address* and *personal property* entities had precision and recall around 0.5. One reason *address* got such low performance is that the researcher later found that there was an error in the lexicons. For the *personal property* entity the low performance was expected since it can include a lot of different objects and is therefore hard to identity.

Yang and Chow [7] also saw that there was a need for a more efficient way to extract valuable information from large amounts of text in investigations. They saw that the methods used in investigations, like keyword searches, have some downsides. To do a keyword search you first need to know what keywords to search for and in addition it might be more matches to the keywords than the investigators are able to efficiently go through. They therefore looked at creating an information extraction system to be used in investigations, which can extract knowledge and data from text. It consists of two parts. The first part is NER, which identifies *people*, *organizations* and *locations*. The next part is relation extraction. In investigations there is often a large amount of data, and it can then be valuable to find relations between the found named entities.

The NER part combines a rule-based and a machine learning approach to NER by using the results from both approaches. The machine learning part is a CRF

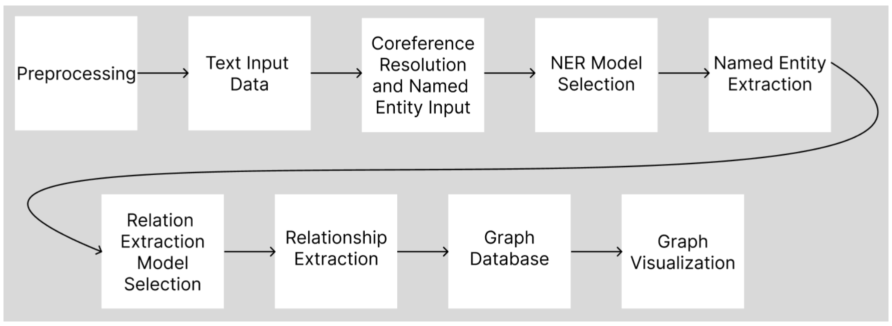


Figure 3.1: Illustration of the information extraction system from Rodrigues *et al.* [39].

model. The relation extraction phase of the system is a modified Apriori algorithm that finds relations in data. To test the system Yang and Chow [7] annotated a cleaned dataset created by Klimt and Yang [38] which consists of about 200,000 emails. The NER part of the system got a F1-score of over 0.82 for all the entity types, and the *organization* entity got the highest F1-score with 0.91.

Other researchers have also looked at using NER in an information extraction system for investigations, like Rodrigues *et al.* [39]. They propose an information extraction system to be used in investigations to analyze text and extract information from text, that consists of 9 steps. An illustration of the 9 steps is provided in Figure 3.1. Step 5 in the system is NER. The system has multiple NER models and which one to use in each case is chosen in step 4. The models are trained on different languages and are able to identify different entity types. There is also a multilingual model. All the models are based on BERT. After NER is performed in step 5, the extracted entities will be used to create a graph of relations. Rodrigues *et al.* [39] trained multiple NER models. In their experiments they trained models in English and Portuguese, and trained models on different datasets and on different BERT variations. The best F1-score they got on English was 0.91 and on Portuguese it was 0.9. When Devlin *et al.* [17] did experiments with NER on the BERT model they achieved F1-scores of around 0.95. Getting a score of 0.91 on a domain specific NER model is therefore quite good.

There has also been proposed a NER model for crime by Shabat *et al.* [40]. They saw that there can be a lot of interesting and valuable information in media channels, which could result in a lot of text in investigations that is time consuming and difficult to analyze manually. They therefore proposed a crime domain specific NER system. To create the dataset to train their model they used manually annotated data from the Malaysian National News Agency (BERNAMA). The articles that regarded crimes were annotated with the entities *weapon*, *location* and *nationality*. To test their dataset they used both a SVM classifier and a Naïve Bayes algorithm and did 10-fold cross-validation on both. The SVM performed best and got a F1-score of 0.91 for *weapons*, 0.96 for *nationality* and 0.89 for *location*. SVM is better for datasets with a lot of features and will therefore probably work better than the Naïve Bayes algorithm with this type of problem [40].

Wu *et al.* [9] saw that a lot of crimes were planned using social media and looked at using Chinese NER to detect this and alert the law enforcement before the crime occurs. The corpus they used had a normal part and a specialized crime part. They used four entity types, *person*, *location*, *organization* and *crime* and tagged the corpus with the IOB2 tagging style. They used a LSTM-CRF model to do experiments on the created corpus. This is a model that combines a LSTM model and CRF model. The LSTM model is a type of neural network that works especially well for sequential data, like a sentence. The CRF part is an effective way to label serialized data. The LSTM-CRF model was in these experiments used to find the probability of the target classes.

The results Wu *et al.* [9] achieved were quite good. When they tested on just the normal part of the corpus they got a precision of 0.904, a recall of 0.863 and a F1-score of 0.883. When also testing on the crime part they got a little lower score. The precision was then 0.876, the recall 0.832 and the F1-score 0.852. The reason for this difference is that the normal part of the corpus is a lot bigger than the crime part of the corpus. The crime part on the other hand is smaller, and has therefore lower performance. Another problem is that the model is trained only on the one crime entity and not other crime-related entities. This might lead to text related to crime not being identified.

3.2 Norwegian Named Entity Recognition

Even though Norwegian is a low resource language, which means that there have not been done as much research on NLP in this language and that not as much data exists to do research on, Norwegian NER models have been developed [41]. Jørgensen *et al.* [10] created the first publicly available dataset for Norwegian NER, called NorNE. It is based on NDT, where named entity annotations are added on top of the existing dataset. The NorNE dataset consists of 8 different entity types:

- Person (PER)
- Organization (ORG)
- Location (LOC)
- Geo-Political Entity (GPE)
  The geo-political entity consists of two categories:
  - Geo-Political Entity with locative sense (GPE_LOC)
  - Geo-Political Entity with organizational sense (GPE_ORG)
- Product (PROD)
- Event (EVT)
- Derived (DRV)

The NorNE dataset was annotated by two linguists, who had to agree on all annotations. The linguists first did a round of annotation to train before they annotated the dataset properly. Needing to agree on all annotations they had to discuss disagreements, but they had a F1-score of 0.915 on initial agreements. The entities they disagreed the most on was the *GPE* entities. They had a hard time to distinguish between the the *GPE_LOC* and *GPE_ORG* entities and between the *LOC* and *ORG* entities and the two *GPE* categories. This led to Jørgensen *et al.* [10] doing experiments on different label sets. They did experiments on three different label sets, one with all 8 entity types, one with just the general *GPE* type and one without the *GPE* entity types.

Jørgensen *et al.* [10] trained NER models on both the written standards of Norwegian, Bokmål and Nynorsk. They trained one model on just Bokmål text, one on just Nynorsk text and one on both.

To test the NER dataset they created, Jørgensen *et al.* [10] used a framework which combines a Convolutional Neural Network (CNN), a Bidirectional Long Short-Term Memory (BiLSTM) and a CRF inference layer, called NCRF++. After running experiments on different label sets and also on different annotations styles, they found that they achieved the best results when using the IOBES annotation style in combination with only using the *GPE* entity, and not the *GPE_ORG* and *GPE_LOC* entities. They then got a F1-score of 0.923. All the combinations of label sets and annotation styles got a F1-score over 0.9. Jørgensen *et al.* [10] also did experiments to compare the Bokmål, Nynorsk and the joint model. The experiments show that the model trained on both Bokmål and Nynorsk performs very well on both Bokmål and Nynorsk text, and this means that it is only necessary to maintain one joint Norwegian model and not one for Bokmål and one for Nynorsk.

Other researchers have also looked into a Norwegian NER model. Johansen [42] also created a NER model and looked at combining a Bokmål and Nynorsk model. Johansen [42] used the entity types *location*, *miscellaneous*, *organizations* and *people*. A converted version of the NDT [26] was used and proper nouns were tagged with the corresponding entity. Only tagging words already classified as proper nouns in the NDT might lead to some words are not being tagged as the correct entity. The IOBES annotation style was used to tag the proper nouns. When doing experiments on the created dataset Johansen [42] achieved a F1-score of 0.867 on the combined model. The reason this score is lower than what Jørgensen *et al.* [10] achieved, in addition to only tagging already classified proper nouns, could be that the dataset is tagged by only one annotator one time, while Jørgensen *et al.* [10] used two annotators in two rounds. It could also be because other and fewer entity types were used.

Johannessen *et al.* [43] have worked on NER models for the mainland scandinavian languages, Norwegian, Swedish and Danish. Another Norwegian corpus annotated with NER was created, based mainly on magazines and news, but it was never made publicly available due to copyright restrictions.

Chapter 4

Methodology

This chapters goes over the methodology of the project, which includes making decisions, collecting and annotating data and fine-tuning models. Figure 4.1 shows the overall methodology.

4.1 Dataset

4.1.1 Entity types

In order to create a NER model we needed a dataset to train it on, but before doing that we had to decide on which entity types to use. Other Norwegian NER models only have a general set of entity types, but we also needed some investigative domain specific entity types. Even though the domain specific entity types are most important and will give most value in an investigation, the general entity types, like *person*, *location* and *organization*, can also be valuable. In the NER model Jørgensen *et al.* [10] created and published they used the general entity types *person*, *organization*, *location*, *geo-political entity with locative sense*, *geo-political entity with organizational sense*, *product*, *event* and *derived*. We used their general entity types as a starting point. Jørgensen *et al.* [10] did experiments with omitting the *geo-political* entity type and just using the *location* and *organization* types, and that gave better performance than using the *geo-political* entity type. In addition we did not see the need for a *geo-political* entity type when using NER for investigative purposes. This is because in an investigation it is more important to know

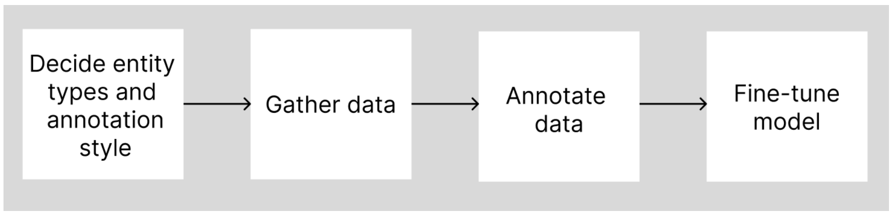


Figure 4.1: Methodology

if an entity is a *location* or an *organization*. We therefore omitted the *geo-politcal* entity type in our work. We did also not use the *derived* entity type, which are not named entities, but are derived from named entities. The *derived* entity type is not often used and it will give little additional value to investigators since this entity type does not say anything about the phrases in this category. We therefore used the general entity types:

Person (PER) People or fictional charachters.

Location (LOC) All types of geographical locations. Cities, addresses, hospitals...

Organization (ORG) Groups of people, like firms, musical groups, organizations...

Product (PROD) This includes products, but also tv shows, laws and other produced objects.

Event (EVT) Events, like wars and festivals. Criminal cases, like the "NOKAS robbery", is also categorized as an event.

When it comes do the domain specific entity types we mainly looked at related work to see which entity types could be useful. When creating their NER model for investigative purposes, Chau *et al.* [8] used the domain specific entity types *vehicle*, *narcotic drug* and *personal property*. To figure out which entity types to use they talked to real investigators and analysts, and they therefor got a good insight in what entity types are useful in investigations. We have therefore chosen to use all those entities. In their research, Shabat *et al.* [40] used the entity types *nationality*, *weapons* and *location*. While the *weapon* entity can be very useful in investigations, we have chosen to not use the *nationality* entity due to ethical and privacy concerns. Ku *et al.* [44] also identified *weapons* in their work.

An entity type that has not been seen in any research, but that we still decided to use is an entity type called *relation*. The *relation* entity type is used on words that describe relations between people, like brother, mother, girlfriend etc. When looking at the data that was going to be used in the dataset, we early saw that a lot of people were mentioned only in the context of their relation to someone else, like "Her mother found the knife under the bed". The mother can here be very useful for investigators. In evidence, like text messages, investigators might often encounter text where people are mentioned only by their relations to someone. The *relation* entity type can therefore be very useful in investigations.

The domain specific entity types we used was then:

Vehicle (VHC) Vehicle includes cars and other means of transport, like airplanes and boats. Both vehicles that are named with brand and model, and specific types of vehicles, like getaway car, is annotated in this category.

Narcotic drug (DRG) Every type of narcotic drug.

Personal property (PRT) Every kind of object that is possessed by someone and does not belong to any of the other entity types.

Weapon (WPN) Both weapon names and weapon types is included here. Every-day objects that are used as weapons, like knife, does also belong to this entity type.

Relation (REL) Every relation between individual, both social and family relations.

During the process of choosing entity types we were able to have a discussion with an investigator about what entity types could be valuable and how a NER model could be used in investigations. This discussion confirmed the entity types we had already looked at. One key takeaway from the conversation is that the cases the police handles are very different from each other. There is no commonalities that will be important in every type of case and what information is valuable for the investigators varies. It is therefore important to make a model that is as general as possible. In addition the investigator told us that the *people* entity type are always interesting in an investigation. This all means that the general entity types we have chosen to include, especially *person names*, could be valuable in investigations. *Personal property* was the entity type we were most uncertain if would add any value to the investigators. The investigator we talked to pointed out that *weapons* and *vehicles* might not be interesting in every case. In some cases a jacket or a pen might be of importance. We therefore chose to also use the *personal property* entity types to be able to identify any object that may be of interest.

4.1.2 Annotation Style

Before starting to work on the data we needed to decide on which annotation style to use. Jørgensen *et al.* [10] did experiments with different annotation styles. They acheived the best performance for the entity types we are going to use with the IOBS annotation style, even though there was not much difference in the performance. The IOBS style acheived a F1-score of 0.919, while the IOB style got 0.91. Even though IOBS acheived the best performance, we decided to use the IOB2 annotation style. This is because the NorNE dataset Jørgensen *et al.* [10] published is annotated with the IOB2 style. Annotating our whole dataset manually would be very time consuming and we therefore used the NER model Jørgensen *et al.* [10] trained to annotate the general entity types for us, before we manually verified the annotations. It was therefore convenient to use the same annotation style as their model and we therefore chose the IOB2 style.

4.1.3 Creating and Annotating The Dataset

We wanted to be able to publish and share the dataset and model we were going to create. Real investigation data are not publicly available and contain a lot of sensitive and personal data. We could therefore not use real investigation data, even though that would have given us a better and more accurate model. The

Table 4.1: Distribution of crime types in the collected articles

Crime Type	Number of articles
Murder	77
Serial Killers	14
Drugs	12
Terror	18
Disappearances	8
Robbery	15
Other	35
Total	179

model would then have been trained on the same kind of data that it was going to be used on and that would give a model that performed better. Since this is not possible for our project we needed to find other data. To look into if it is possible to train a Norwegian NER model for investigative purposes we need domain specific data that contains the entity types we are going to use in the model. We therefore chose to use Norwegian Wikipedia articles regarding or related to crime. This is publicly available, but still domain specific.

Before collecting the data we sent a request to Sikt to make sure everything is done according to GDPR. The application was approved, but there are significant limitations to how we can share the data. The assessment we got on our application can be seen in Appendix A. The data we collect contains personal information and we can therefore not distribute the data as we want. The collected data can not be published or shared, even though it is publicly available. This means that we can not share and publish the dataset and the model as we originally wanted, but we still had to make the model with Wikipedia data, since we do not have access to real investigation data.

To find relevant Norwegian Wikipedia articles related to crime we first found articles about well known cases in Norway, like "NOKAS-ranet" (NOKAS robbery), "Orderud-saken" (Orderud case) and "Terrorangrepene i Norge 2011" (2011 Norway attacks). From these articles we followed hyperlinks and category references to locate other articles. We collected articles about cases in both Norway and other countries, as long as the article was writtem in Norwegian. Most articles in our dataset is about murders, but there are also articles about other crimes, like robbery, terror, drugs and disappearances. In total we collected 179 articles. The distribution of the crime types in the articles are shown in Table 4.1. The other category consists of all the articles that does not fit into any of the other categories and includes among others articles regarding espionage, abuse and fraud.

To efficiently collect the Wikipedia articles we used Requests[1] and Beautiful

[1]https://pypi.org/project/requests/

Code listing 4.1: Script to collect and structure Wikipedia files

```
article_file = open('articles.txt', 'r')

articles = article_file.read()

article_list = articles.split('\n')

for article in article_list:
        if not article == '':
                page = requests.get(article)
                soup = BeautifulSoup(page.content, 'lxml')
                p_list = soup.find_all('p')
                full_text = ''
                for p in p_list:
                        p_text = p.get_text().replace('\n', '')
                        p_text = re.sub('\[[0-9]+\]', '', p_text)
                        full_text += '␣' + p_text
                with open('articles.csv', 'a', newline='') as file:
                        writer = csv.writer(file)
                        writer.writerow([soup.find('h1').get_text(), full_text])
```

Soup,[2] a webscraping library. This made it easy to efficiently extract and structure the content from the Wikipedia articles. Some parts of the Wikipedia articles are not relevant for our work and Beautiful Soup allowed us to extract only the relevant text and not headlines, references and footnotes. We created a script that first uses Requests to scrape the Wikipedia articles. The links to the Wikipedia articles that were going to be used were listed in a text document that our script read from. Beautiful Soup was then used to find all the paragraphs of the Wikipedia articles. The script removes the citations in the text and finally adds all the articles to a csv file. Code listing 4.1 shows an excerpt of this script and Appendix B shows the whole script.

To be able to fine-tune a NER model we needed to annotate the dataset with the correct labels. To do this manually would be very time consuming. Jørgensen *et al.* [10] have published a Norwegian NER model, which we therefore used to do some of the annotating for us. Their NER model, nb-bert-base-ner,[3] uses the IOB2 annotating style and annotates the general entity types we use. The model also annotates the *geo-political* entity type and the *derived* entity type which we are not going to use, but we changed those when manually annotating later.

We created a script, based on a tutorial on Huggingface[4] that reads the csv files with the articles and sends each sentence in the articles through the nb-bert-base-ner model. Code Listing 4.2 shows an excerpt of the script and Appendix C provides the whole script. The script also structures each sentence and writes the sentence and the annotations to a json file. It creates one file per article. The

[2]https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[3]https://huggingface.co/NbAiLab/nb-bert-base-ner
[4]https://huggingface.co/NbAiLab/nb-bert-base-ner

Code listing 4.2: Script to do initial annotation and structure data in json files

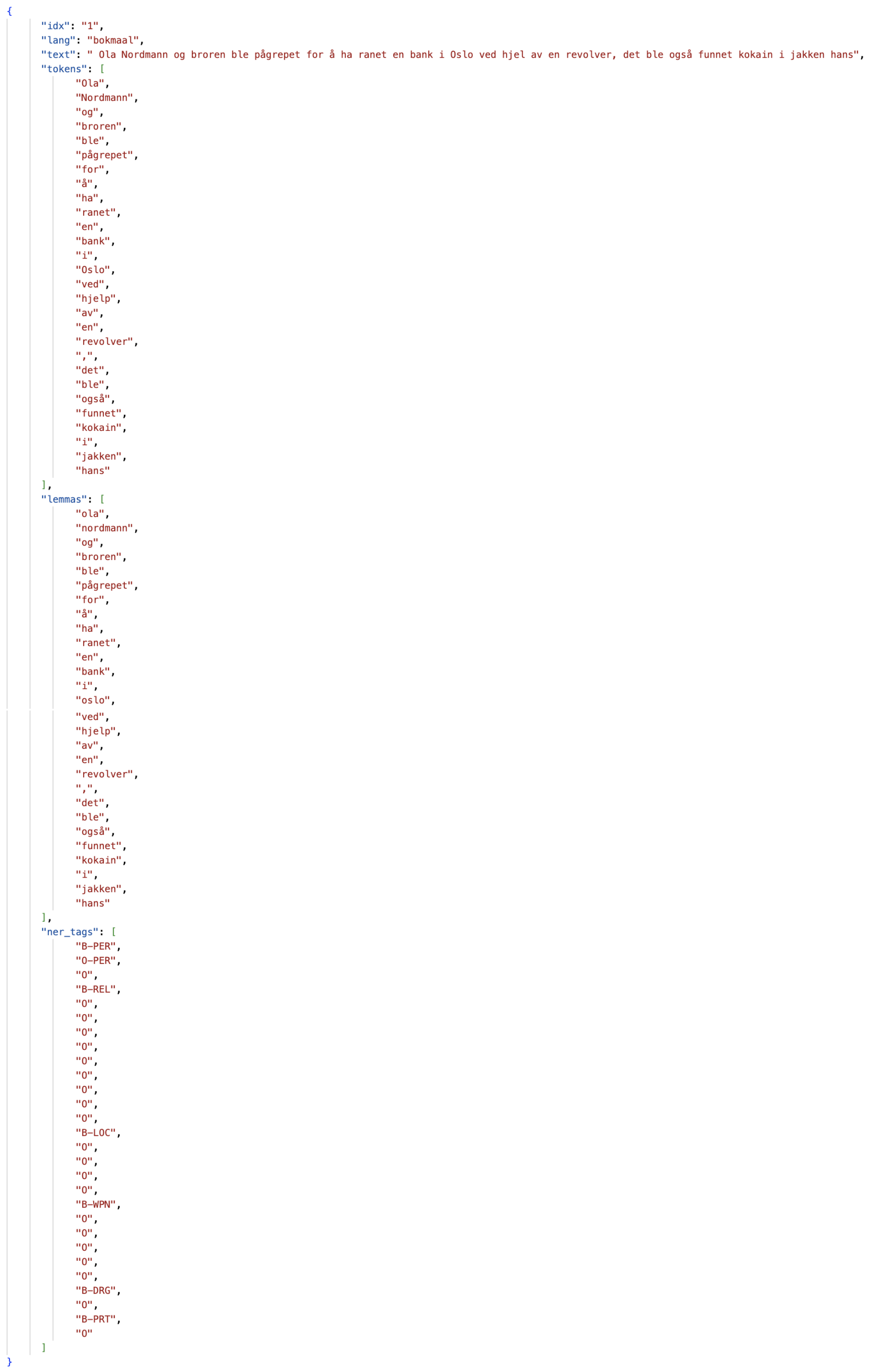
```
tokenizer = AutoTokenizer.from_pretrained("NbAiLab/nb-bert-base-ner",
                 model_max_length=512)
model = AutoModelForTokenClassification.from_pretrained("NbAiLab/nb-bert-base-ner")

ner = pipeline("ner", model=model, tokenizer=tokenizer)

with open("beautifulSoup/articles.csv", "r") as file:
   csvreader = csv.reader(file)
   next(csvreader)
   for row in csvreader:
      file = open('files/'+row[0]+'.json', 'a')
      sentences = row[1].split('.')
      for sentence in sentences:
         if len(sentence) <= 512:
            ner_result = ner(sentence)
            # [...creating the sentence object...]
            ner_tags = []
            for word in tokens:
               ner_match = list(filter(lambda j: j['word'] == word, ner_result))
               if len(ner_match) > 0:
                  ner_tags.append(ner_match[0]['entity'])
               else:
                  ner_tags.append("O")
            sentence_object["ner_tags"] = ner_tags
            json_object = json.dumps(sentence_object, indent=6, ensure_ascii=False)
            file.write(json_object)
```

structure of the json file is based on the structure of the NorNE dataset[5] that Jørgensen *et al.* [10] created and published which the nb-bert-base-ner is fine-tuned on. We did not do a lot of preprocessing on the data. This is because it will not harm the model if there is some noise in the data, this will just be labeled with the "O" tag, outside named entity. In addition there will be noise in real data, so the model will be more robust if it is trained on noisy data.

After the nb-bert-base-ner model annotated some of the named entities for us, we went through the files manually. This had two purposes. The first purpose was to check that all the annotations from the nb-bert-base-ner was correct and to add annotations that were missing. Even though this model performs well it is not perfect. Foreign names and places are not as easyily detected as Norwegian names. We therefore had to add some of the general entity annotations manually. We also had to change all the *geo-political* annotations to *location* and *organization* annotations, and remove or replace the *derived* annotations. The second purpose was to add the domain specific entity types. All these annotations needed to be added manually. One annotator went through all the files one time and checked and labeled all the named entities. The distribution of named entities in the dataset can be seen in Table 4.2. An example of an annotated sentence, as in the dataset, can be seen in Figure 4.2.

When fine-tuning the NER model we needed a training dataset and a valid-

[5]https://huggingface.co/datasets/NbAiLab/norne



```
{
    "idx": "1",
    "lang": "bokmaal",
    "text": " Ola Nordmann og broren ble pågrepet for å ha ranet en bank i Oslo ved hjel av en revolver, det ble også funnet kokain i jakken hans",
    "tokens": [
        "Ola",
        "Nordmann",
        "og",
        "broren",
        "ble",
        "pågrepet",
        "for",
        "å",
        "ha",
        "ranet",
        "en",
        "bank",
        "i",
        "Oslo",
        "ved",
        "hjelp",
        "av",
        "en",
        "revolver",
        ",",
        "det",
        "ble",
        "også",
        "funnet",
        "kokain",
        "i",
        "jakken",
        "hans"
    ],
    "lemmas": [
        "ola",
        "nordmann",
        "og",
        "broren",
        "ble",
        "pågrepet",
        "for",
        "å",
        "ha",
        "ranet",
        "en",
        "bank",
        "i",
        "oslo",
        "ved",
        "hjelp",
        "av",
        "en",
        "revolver",
        ",",
        "det",
        "ble",
        "også",
        "funnet",
        "kokain",
        "i",
        "jakken",
        "hans"
    ],
    "ner_tags": [
        "B-PER",
        "O-PER",
        "O",
        "B-REL",
        "O",
        "O",
        "O",
        "O",
        "O",
        "O",
        "O",
        "O",
        "O",
        "B-LOC",
        "O",
        "O",
        "O",
        "O",
        "B-WPN",
        "O",
        "O",
        "O",
        "O",
        "O",
        "B-DRG",
        "O",
        "B-PRT",
        "O"
    ]
}
```

Figure 4.2: Annotation of the sentence "Ola Nordmann and his brother were arrested for robbing a bank in Oslo with a revolver, it was also found cocain in his jacket"

Table 4.2: Distribution of entity types in the full dataset

Entity Type	Tokens
PER	7779
LOC	3499
ORG	3775
PROD	929
EVT	548
VHC	122
DRG	51
PRT	402
WPN	442
REL	824

Table 4.3: Dataset size

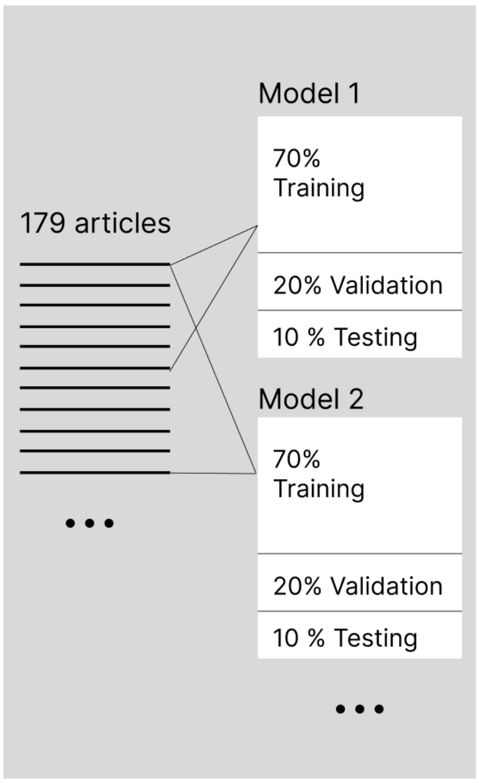
Model	Articles	Sentences	Named Entity Tokens	Domain Specific Named Entity Tokens
Model 1	41	2890	5195	302
Model 2	75	4806	8930	926
Model 3	107	6769	12723	1243
Model 4	145	8444	15761	1533
Model 5	179	9874	18371	1841

ation dataset. We also wanted a separate test dataset so that we could test the model on unseen data. We therefore created a script that takes all the annotated files and split them up in a train, validation and test dataset. The script also joins all the files for each dataset. Code Listing 4.3 shows the main parts of this script. Appendix D provides the whole script. After running this script we therefore ended up with one training file, one validation file and one test file. 70% of the original dataset was used for training, 20% for validation and 10% was used for testing.

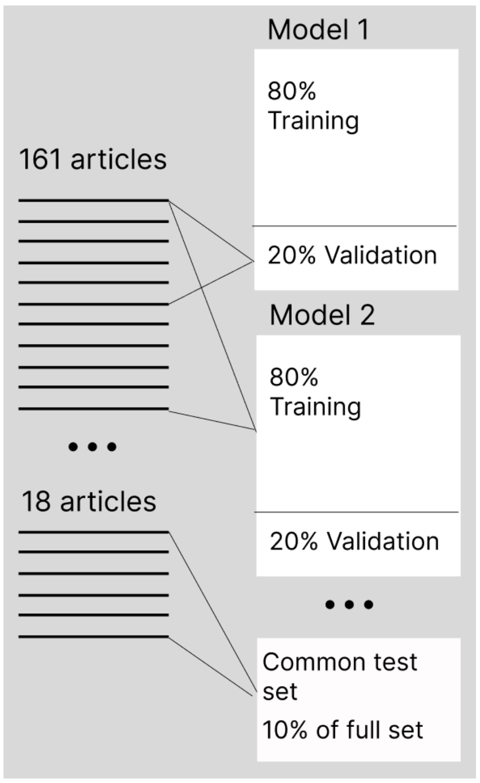
To see how our model improved with more training data, we did not fine-tune our model with all the collected data initially. We scraped and annotated just a small part of all the Wikipedia data first and then fine-tuned a model with that data. Then we scraped and annotated some more articles and included them in the dataset before we fine-tuned a new model. This was done 5 times and the amount of data each of these 5 models were trained on can be seen in Table 4.3. An illustration of how we trained multiple models with different amounts of data and how we split the data into training, validation and testing sets is provided in Figure 4.3a.

Code listing 4.3: Script to split the annotated data in train, validation and test files

```
files = os.listdir('files')
random.seed(42)
random.shuffle(files)

train_files, validation_files, test_files = np.split(files,
                       [int(len(files)*0.7), int(len(files)*0.9)])
overview_file = open('data/overview.txt', 'w')

def list_in_overview(files, title):
   overview_file.write('\n'+title+'\n')
   for file in files:
      overview_file.write(file+'\n')

def merge_files(files, file_name):
   all_sentences = []
   for file in files:
      file = open('files/'+file, 'r')
      all_sentences.extend(json.load(file))
   merged_file = open('data/'+file_name+'.json', 'w')
   json_objects = json.dumps(all_sentences, indent=6, ensure_ascii=False)
   merged_file.write(json_objects)
```



(a) First round of training



(b) Second round of training

Figure 4.3: Illustrations of the training proccess.

4.2 Fine-tuning Model

We decided that we wanted to fine-tune a pretrained model instead of training a model from scratch. When fine-tuning a pretrained model we could use a smaller dataset than we would have needed to train a model from scratch, which is very fitting for this project. When we are using a smaller dataset, that also means that training the model will take less computational resources and be less time consuming. For the police it would also be beneficial to fine-tune an already pretrained model since they then could achieve good results with less data and in shorter time. They would also need fewer resources.

As mentioned in Section 2.2.2, the Transformer architecture has become very popular for NLP tasks. BERT, mentioned in Section 2.2.3, which are based on the Transformers architecture, have also become very popular and shows promising results. We will therefore fine-tune a BERT based model. The model we used is the *nb-bert-base* model.[6] This model is part of the NoTraM project[7] created by the National Library of Norway. The NoTraM project aims to create a Norwegian corpus and transformer-based models for Norwegian. They have done tests on their *nb-bert-base* model, which performed very good with an F1-score of 0.987 on POS tagging and 0.9 on NER [24].

Huggingface,[8] which is also mentioned in Section 2.2.2 has also become very popular and the model we used is available in the Huggingface repository. We therefore used the Huggingface repository and the Huggingface libraries to fine-tune our model. Huggingface has a script for fine-tuning models for token classification tasks, which includes Named Entity Recognition. We used this script[9] with some minor modifications to fine-tune our model. The same script can also be used to compute the overall performance of the model when tested on the test dataset. The part of the script that does the actual fine-tuning is provided in Code Listing 4.4. The whole script is provided in Appendix E.

When we started to fine-tune our models we first fine-tuned the 5 models described in Table 4.3. We did this to be able to compare how the models performs with different amounts of training data. The randomly done split into training, validation and test data lead to these models being tested on different test data and the comparison was therefore not that accurate. After training these models, we therefore wanted to repeat the training and testing, but this time with the same and unseen data in the test set for all 5 models. We did this by using the test set from Model 5, which was randomly chosen, and deleted the data from this test set from the training and validation sets in the first 4 models. We then had a large test set with unseen data common for all 5 models. When we split the data this time we used 80% of the remaining data for training and 20% for validation. After training these models we tested all five models on the large unseen test set

[6]https://huggingface.co/NbAiLab/nb-bert-base

[7]https://github.com/NBAiLab/notram

[8]https://huggingface.co

[9]https://github.com/huggingface/transformers/tree/main/examples/pytorch/token-classification

Code listing 4.4: Excerpt from script that fine-tunes the model

```
# The original file is modified
# Copyright 2020 The HuggingFace Team All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
    # Training
if training_args.do_train:
    checkpoint = None
    if training_args.resume_from_checkpoint is not None:
        checkpoint = training_args.resume_from_checkpoint
    elif last_checkpoint is not None:
        checkpoint = last_checkpoint
    train_result = trainer.train(resume_from_checkpoint=checkpoint)
    metrics = train_result.metrics
    trainer.save_model()  # Saves the tokenizer too for easy upload

    max_train_samples = (
        data_args.max_train_samples if data_args.max_train_samples is not
          None else len(train_dataset)
    )
    metrics["train_samples"] = min(max_train_samples, len(train_dataset))

    trainer.log_metrics("train", metrics)
    trainer.save_metrics("train", metrics)
    trainer.save_state()
```

and obtained comparable results. An illustration of how this was done can be seen in Figure 4.3b.

After annotating all the data and fine-tuning the last model on all the data we did some brief experiments to discover the optimal hyperparameters, such as number of epochs and learning rate. Epochs is the number of times the algorithm should go through the training set and train the model. This is important to not over- or underfit the model. All the models up to this were trained with 3 epochs as this was the default in the script we used. Deciding on the optimal number of epochs will give the best possible performance for the model and make it more fit for encountering unseen data. Devlin *et al.* [17] states that when fine-tuning BERT the optimal number of epochs is 2, 3 or 4 for a lot of different tasks. We therefore chose to train models with epochs from 1 to 10, to see how the F1-score and loss of the models changes. These models were trained, validated and tested with all the 179 articles.

The learning rate tells how much the model should change in each iteration. Devlin *et al.* [17] suggests the learning rates 5e-5, 3e-5 and 2e-5. The previous models were all trained with a learning rate of 5e-5, since that was the default in the script we used. After concluding on the optimal number of epochs we trained two new models with the two other suggested learning rates and the optimal number of epochs.

For all the models we trained we tested it on an unseen test dataset and computed the precision, recall and F1 score. In addition to the overall performance we wanted to see how the models performed on the different entity types. This showed us which entity types the model recognized easiest and what kind of data were missing from the dataset. To do this we used seqeval, which is a Python framework that evaluated the performance of different labels in token classification tasks [45]. We used the Evaluate library[10][11] from Huggingface to compute the performance. The script we used to compute the performance is provided in Appendix F.

[10]https://huggingface.co/spaces/evaluate-metric/seqeval
[11]https://huggingface.co/docs/evaluate/index

Chapter 5

Results

The dataset we created was tested in a couple of different ways. First, all the models were tested as they were created and they were then tested on different data and different amounts of data. The results from these tests can be seen in Table 5.1 and 5.2.

When training a model with more data, the performance should in theory be better, but Table 5.1 and 5.2 shows that the performance of the models varies and sometimes are lower with more data. The data in the test set were in this case quite small for the first models and there were few domain specific named entities. There were for instance no occurrences of *narcotic drug* in the first test set and only 2 instances of *vehicles* for the second model. This, in addition to the models being tested on different data makes this approach not suitable for comparing the performance of the models. Figure 5.3 shows the number of files each model was initially trained, validated and tested on.

After training new models and leaving out a larger common unseen test dataset for all the models we got results better suited for comparison. All the models were then tested on 18 articles with a total of 1153 sentences. The test data included 2207 named entities, where 221 were domain specific. The F1-score from these tests on all the different entities can be seen in Table 5.4 and 5.5, and the precision, recall and F1 score of each model can be seen in Table 5.6.

In Table 5.4, 5.5 and Table 5.6 it can be seen that the performance of the

Table 5.1: F1-score of models with different amount of training and different test data

Model	PER	LOC	ORG	PROD	EVT
Model 1	1.0	0.927	0.896	0.5	0.75
Model 2	0.995	0.935	0.915	0.455	0.80
Model 3	0.959	0.882	0.794	0.833	0.870
Model 4	0.983	0.943	0.899	0.818	0.943
Model 5	0.949	0.935	0.842	0.667	0.793

Table 5.2: F1-score of models with different amount of training data and different test data

Model	VHC	DRG	PRT	WPN	REL	Total
Model 1	1.0	-	0.5	1.0	1.0	0.916
Model 2	0.666	0.0	0.333	1.0	0.933	0.921
Model 3	1.0	0.615	0.5	1.0	0.930	0.883
Model 4	0.970	0.6	0.647	0.927	0.955	0.939
Model 5	0.973	0.640	0.693	0.851	0.930	0.906

Table 5.3: The distribution of files in the different models

Model	Training	Validation	Test
Model 1	28	8	5
Model 2	52	15	8
Model 3	74	22	11
Model 4	101	29	15
Model 5	125	36	18

Table 5.4: F1-score of models with different amount of training data, but a common test set. All trained with 3 epochs

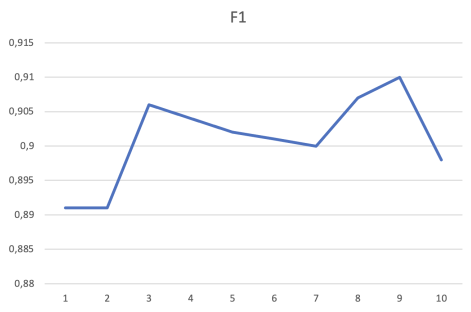
Model	PER	LOC	ORG	PROD	EVT
Model 1	0.938	0.917	0.782	494	0.737
Model 2	0.943	0.943	0.822	0.659	0.641
Model 3	0.948	0.930	0.815	0.6	0.852
Model 4	0.940	0.936	0.813	0.602	0.719
Model 5	0.949	0.935	0.842	0.667	0.793

Table 5.5: F1-score of models with different amount of training data, but a common test set. All trained with 3 epochs

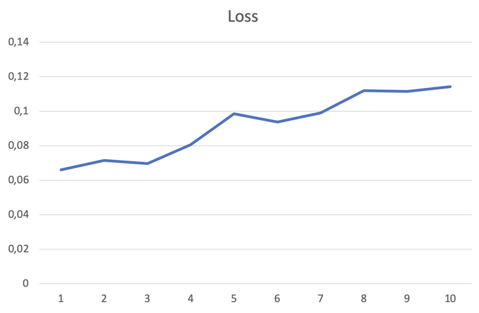
Model	VHC	DRG	PRT	WPN	REL	Total
Model 1	0.944	0.667	0.575	0.875	0.907	0.876
Model 2	0.971	0.692	0.548	0.845	0.899	0.893
Model 3	0.971	0.786	0.619	0.907	0.928	0.897
Model 4	0.941	0.640	0.676	0.863	0.923	0.893
Model 5	0.973	0.640	0.693	0.851	0.930	0.906

Table 5.6: Performance of models with different amount of training data and same test data

Model	Precision	Recall	F1-score
1	0.871	0.881	0.876
2	0.887	0.899	0.893
3	0.887	0.907	0.897
4	0.888	0.898	0.893
5	0.904	0.908	0.906



(a) F1



(b) Loss

Figure 5.1: F1 and loss when testing models trained with different number of epochs

models gets better when more data is used for training.

When doing experiments to find the optimal number of epochs for training we measured the F1-score of the individual entity types, in addition to the overall F1-score, precision, recall and loss. The training, validation and test data for Model 5 from Table 5.4 and 5.5 where used when trying out different epochs. The F1-score of the individual entity types can be seen in Table 5.7 and 5.8, and the overall precision and recall in Table 5.9. The F1-score and loss have been plotted in the graphs in Figure 5.1. This is done so that we easily can see how the F1-score and loss changes when the number of epochs increases. Loss measures the error of the model.

When deciding on a optimal number of epochs the F1-score and loss measures are reasonable to use. We see in Figure 5.1a, that shows the F1-score on the test data, that this score is best with 3, 8 and 9 epochs. The same can be seen in Table 5.7 and 5.8, and Table 5.9, the performance is best with 3, 8 and 9 epoch. When looking at the loss on the test data, which shows the error of the model, in Figure 5.1b, we see that the loss goes up when using more than 3 epochs. We therefore decided that the optimal number of epochs for us to use is 3. This also fits the suggestion given by the creators of BERT [17].

After deciding on using 3 epochs we did experiments with the learning rate. The performance of the models trained with different learning rates can be seen in Table 5.10. The table shows that there is not a lot of difference, but that the best learning rate is 5e-5. This means that the optimal model for us to use is the model that is created with all the 179 articles, minus the test data, trained with 3 epochs and a learning rate of 5e-5. This model is trained on 125 articles and validated with 36 articles. Training this model took 2 minutes and 24 seconds. Detailed performance of this model can be seen in Table 5.11. When doing experiments on this model it was tested on 18 articles.

The performance varies a lot between the different entity types. The entity type with the lowest performance is *narcotic drug*. The reason for this is that this entity type has a lot fewer occurrences in the dataset than the others, with only 51 tokens. *Personal property* also has low performance, even though there are a lot of occurrences of this entity type in the dataset. This comes from that the *personal property* entity types includes a lot of different phrases where there is no coherence in the syntax and semantics, it could be any kind of object that is owned by someone. This makes it difficult to annotate and also difficult for the model to detect. The *product* entity type has even more occurrences in the dataset, but also have low performance, due to the same reasons. For *product* there is also no coherence in syntax and semantic.

Vehicle has the second fewest occurrences in the dataset, but still has good performance. This could be because there is some coherence in the syntax, context or semantic for these entities or because some words is repeated in many of the entity phrases. It would be reasonable to believe that more occurrences in the dataset would lead to better performance, but we can see from Table 4.2 and Table 5.11 that this is not the case for multiple of the entity types. This means that how well the model recognizes different entity types also depends on other factors. These factors could be syntax, context and semantic, in addition to repetition of specific words. For instance the *relation* entity type only includes specific words, like "mor" (mother), "sønn" (son) and "kjæreste" (boyfriend/girlfriend). This makes it easier for the model to recognize these entities.

Table 5.7: F1-score of model 5 with different numbers of epochs

Epochs	PER	LOC	ORG	PROD	EVT
1	0.943	0.916	0.816	0.575	0.746
2	0.938	0.922	0.817	0.563	0.754
3	0.949	0.935	0.842	0.667	0.793
4	0.950	0.937	0.844	0.595	0.767
5	0.948	0.928	0.835	0.643	0.780
6	0.941	0.935	0.840	0.652	0.767
7	0.951	0.927	0.816	0.625	0.842
8	0.950	0.935	0.849	0.659	0.721
9	0.949	0.945	0.852	0.650	0.738
10	0.947	0.933	0.817	0.587	0.697

Table 5.8: F1-score of model 5 with different numbers of epochs

Epochs	VHC	DRG	PRT	WPN	REL	Total
1	0.919	0.615	0.675	0.860	0.933	0.891
2	0.973	0.640	0.700	0.907	0.938	0.891
3	0.973	0.640	0.693	0.851	0.930	0.906
4	0.944	0.640	0.684	0.891	0.914	0.904
5	0.941	0.640	0.676	0.882	0.920	0.902
6	0.941	0.640	0.693	0.882	0.927	0.901
7	0.919	0.692	0.623	0.882	0.920	0.90
8	0.944	0.640	0.687	0.894	0.927	0.907
9	0.944	0.786	0.711	0.894	0.919	0.910
10	0.909	0.692	0.706	0.872	0.931	0.898

Table 5.9: Performance of model 5 with different numbers of epochs

Epochs	Precision	Recall	F1-score
1	0.882	0.90	0.891
2	0.881	0.902	0.891
3	0.904	0.908	0.906
4	0.899	0.909	0.904
5	0.901	0.902	0.902
6	0.895	0.907	0.901
7	0.896	0.902	0.90
8	0.906	0.908	0.907
9	0.907	0.913	0.910
10	0.894	0.902	0.898

Table 5.10: Performance of model 5 with different learning rate

Learning Rate	Precision	Recall	F1-score	Loss
5e-5	0.904	0.908	0.906	0.070
3e-5	0.898	0.904	0.901	0.075
2e-5	0.892	0.910	0.901	0.076

Table 5.11: Precision, recall and F1-score of each entity type with optimal hyper-parameters, model 5 with 3 epochs and a learning rate of 5e-5

Entity Type	Precision	Recall	F1-score
PER	0.966	0.932	0.949
LOC	0.925	0.944	0.935
ORG	0.810	0.876	0.842
PROD	0.609	0.739	0.667
EVT	0.742	0.852	0.793
VHC	0.947	1.0	0.973
DRG	1.0	0.471	0.640
PRT	0.605	0.813	0.693
WPN	0.909	0.8	0.851
REL	0.952	0.909	0.930
Total	0.904	0.908	0.906

Chapter 6

Discussion

The discussion will answer the research questions, look at the performance of the model, and the dataset and disadvantages with the how the dataset is created. It will also look at limitations with our model and how to deal with other languages.

6.1 Answering Research Question 1: Creating a NER Model

Research question 1, *How can we develop a Norwegian NER model fine-tuned for investigation, with the restriction of using open-source data?*, is partly answered in Chapter 4, which explains how such a model can be created. We have shown that it is possible to create a Norwegian NER model for investigative purposes with open-source data and how it could be done.

6.2 Answering Research Question 2: Performance

We now consider how our research have answered our research questions. Research question 2, *What kind of precision, recall and F1-score could such a model achieve and how does it compare with the state-of-the-art Norwegian NER model trained for general purposes and NER models created for investigative purposes in other languages?*, deals with the performance of the model we have created. The performance of the model is quite good. Chau *et al.* [8] also created a NER model for investigative purposes and used the entity types *person*, *address*, *narcotic drug* and *personal property*. Our model performed better than their model on most of the entity types. A table with comparisons of the F1-score of our model with other models for each entity type is provided in Table 6.1. For the *person* entity they achieved a precision of 0.741 and a recall of 0.734, while we got 0.966 and 0.932. *Address* and *location* are quite similarly entity types, our model would be able to identify addresses as locations, and it is therefore reasonable to compare them. On their model they got a precision of 0.596 and a recall of 0.514, while our model achieved a precision of 0.925 and a recall of 0.944. Our *personal property* entity type did not perform as well as other entity types, but got a precision and recall of

0.605 and 0.813. This was still a lot better than the model Chau *et al.* [8] created, which got 0.468 and 0.478. Our model achieved perfect precision for *narcotic drug*, while the other model only got a precision of 0.779. On the other hand, our model only got a recall of 0.471 compared to 0.779 for the other model. Overall the other model therefore performed better on *narcotic drugs*. One reason for the difference in performance could be the architecture of the models. Our model is a BERT based model, while their system consists of a noun phraser, a finite state machine and a neural network. The number of occurrences of the different entity types probably also affect the performance. Our dataset only contained 51 occurrences of *narcotic drug*, while Chau *et al.* [8] had 252 occurrences in their dataset. On the other hand they only had 62 occurrences of *address*, while we had 3499 occurrences of *location*. From this we can see that the number of occurrences of a entity type in the dataset affect the performance of the model significantly. Generally, the model Chau *et al.* [8] created was trained on a small dataset, so we can see that the size of the dataset affects the performance.

Neither models did perform very well on the *personal property* entity type. In our dataset the *personal property* entity type had around the same number of occurrences as the *weapon* entity type, which got a significant higher score. This points to that the number of occurrences is not the reason for the low performance in this case. The reason that *personal property* has low performance is the indistinct definition of the entity type. The entity type consists of any type of physical object that someone owns and can therefore include a lot of different things, like phones, pens and furniture. Chau *et al.* [8] also reported this. In addition there is no consistency in the words, like capital letters in the start of the word, and little syntactic and semantic similarity. This also made it difficult to annotate this entity type. The annotation of this entity type might therefore be inconsistent and words might not be annotated correctly. That will also affect the performance of the model.

The NER model Wu *et al.* [9] created to fight crime does not have any of the same domain specific entity types as our model, but we can still compare the overall performance. Our model got a precision of 0.904, a recall of 0.908 and a F1-score of 0.906. Their model got a precision of 0.876, a recall of 0.832 and a F1-score of 0.852. Wu *et al.* [9] had a larger dataset, but the reason that our model performed better might be the architecture of the model. Their model were a LSTM-CRF, while we used the new and popular BERT architecture, which has been reported to have very good performance.

Shabat *et al.* [40] trained two NER models to use for investigative purposes that had *weapons* and *location* as entity types. The *weapon* entity type got a F1-score of 0.911 and 0.867, while the *location* entity type got a score of 0.893 and 0.787. For the same entity types our model achieved a F1-score of 0.851 for *weapon* and 0.935 for *location*. Our model performed a little better on *location*, but a bit lower on *weapon*. Shabat *et al.* [40] used a larger dataset, with 500 news articles about crime. They therefore probably have more occurrences of *weapons* and were able to get a better score. For *location* on the other hand, we had a

Table 6.1: F1-score of different models on the different entity types

Entity Type	Our Model	Chau *et al.* [8]	Shabat *et al.* [40] SVM	Shabat *et al.* [40] NB	Yang and Chow [7]	Rodrigues *et al.* [39]	Jørgensen *et al.* [10]
PER	0.949	0.737	-	-	0.88	0.92	0.98
LOC	0.935	0.552[1]	0.893	0.877	0.82	0.946	0.907
ORG	0.842	-	-	-	0.91	0.741	0.901
PROD	0.667	-	-	-	-	-	0.822
EVT	0.793	-	-	-	-	-	0.640
VHC	0.973	-	-	-	-	-	-
DRG	0.640	0.815	-	-	-	-	-
PRT	0.693	0.473	-	-	-	-	-
WPN	0.851	-	0.912	0.867	-	-	-
REL	0.930	-	-	-	-	-	-

[1] This model is only tested on address.

lot of occurrences in our dataset, even though our dataset is quite small, and we therefore were able to get good scores for this entity type.

Our model performs better than the NER part of the information extraction system Yang and Chow [7] created on two of the entity types both models uses. On *person* names our model got a F1-score of 0.949, while their system got 0.88 and for *locations* our model achieved a F1-score of 0.935, while their model got 0.82. For *organizations* our model got a F1-score of 0.842, while their model achieved 0.91. The reason that our model performed worse on *organizations* might be the number of occurrences of this entity type, Yang and Chow [7] used a larger dataset than we did and might therefore have more occurrences of it. The reason our model performed better on the other entity types might be the architecture of the model. Our model is BERT based, which is very popular and shows great promise, while they used a CRF model.

The NER part of the system Rodrigues *et al.* [39] created got quite similar F1-scores as we did for *person* and *location*, *person* a litter lower and *location* a little higher. But for *organization* they achieved significantly lower score. Rodrigues *et al.* [39] also used a BERT based model in their experiments, so the architecture of the models is not the reason for the difference here. The reason the models performs so differently on *organization* is therefore probably due to the dataset and the occurrences of this entity type.

Comparing our model to the Norwegian NER model created by Jørgensen *et al.* [10], we can see that out model performed very well. Their model, with just general entity types, achieved a F1-score of at best 0.923. This was with another annotation style than we used and with different general entity types. In this model they used a general *geo-political* entity type. The F1-score they achieved when using the same annotation style as we did and the same set of general entity types were 0.91, which is not a lot higher than our F1-score of 0.906. This shows that it is possible to train a NER model for investigative purposes also for Norwegian. Even when we added a lot of new entity types, the performance did not drop very much. The reason for the small drop with new entity types, is that there are fewer occurrences of these entity types in the dataset and it is therefore more difficult to get good performance on those.

When it comes to the different entity types in the model Jørgensen *et al.* [10] created compared to our model we can see in Table 6.1 that our model performs best on some entity types and their performs better on others. Our model performs better on *location* and *event,* while their models performs best on *person*, *organization* and *product*. The reason for this could be the difference in the distribution of the different entity types in the dataset. Jørgensen *et al.* [10] had for instance very few occurrences of the *event* type in their dataset, while we had a lot more, and this can be seen in the comparison.

When the creators of BERT did experiments with NER in English they got a F1-score of 0.95 [17]. Our model have slightly lower performance, but it is not fine-tuned on a lot of data. The results we got shows great promise and with more data and more occurrences of domain specific entities it is possible to create a Norwegian NER model for investigative purposes with very good performance.

To create a good model both precision and recall is important, but in different situations one might be more important than the other. High recall will lead to few false negatives. In a NER system, a high recall means that there are few occurrences of an entity type that is not recognized. It also means that the false positives, phrases recognized as an entity type it does not belong to, will be higher. As Lillis and Scanlon [46] discusses, this is preferable in a investigation context. Missing some important evidence is more harmful than identifying content that is not important.

When looking at the results of our model we can see that for some entity types the precision is higher, and for some entity types the recall is higher, even though there is not much differences. Both the overall precision and recall is quite good, but the recall is a bit higher, which is fitting for a NER model that is going to be used in investigations.

When training a model it can be difficult to know what to do to increase the recall other than training on more data. Increasing the amount of occurrences of the different entity types will increase both the precision and recall. When experimenting with different data and hyperparameters the importance of recall can be kept in mind and a model with the best recall can be chosen.

6.3 Dataset

The dataset we have created is not perfect. We have used Wikipedia articles and we are therefore limited to what exists on Wikipedia in Norwegian and what we were able to find. We used category pages on Wikipedia to find articles about different types of crime, but the distribution of topics in the Wikipedia articles could have been a lot better. We can see from Table 4.1 in Section 4.1.3 that there are most murder cases in our dataset. This is because that is what we found the most of on Wikipedia. Having most murder cases is not an accurate representation of the real world. Murders happen, but it is not the crime that happens most often. There are more robberies and drug cases. Economic crime and fraud are also greatly represented in the society, but our dataset only contains 3 such cases. Cybercrime is also becoming more and more widespread, but we only found 1 such case. When the model is trained on mostly murder cases it will also perform best on murder cases. We want it to perform well on all types of cases, and it should therefore ideally be trained on the same amount of cases of each type or a representative distribution of types.

The most important thing is that the model is trained on enough instances of named entities. We can see in Table 4.2 in Section 4.1.3 that we have a lot of instances of most of the general entity types. For the domain specific entity types on the other hand, we have a lot fewer instances. This affects the performance of these entities as we can see in Table 5.11 in Section 5. *Person* names, with 7779 tokens, performs a lot better than *narcotic drug*, with 52 tokens. We saw during the annotation process that many of the articles related to crime on Wikipedia still did not contain a lot of the domain specific entity types. That made it difficult to create a dataset with many instances of these entity types. The solution then was to annotate a lot of articles to still get a fair amount of named entities. But the amount of *narcotic drug* entities is still way too low.

After consulting the literature on the subject and talking to an investigator we decided on the domain specific entity types *vehicle*, *narcotic drug*, *personal property*, *weapon* and *relation*. In most cases these entity types, in addition to the general ones, will be sufficient. In a murder case the weapon is important and in a robbery the escape vehicle might be important. In other cases, like for instance economic crime and cybercrime, these entities are not as important. In these cases other things might be important, like amounts of money or computer systems, but we do not have any entity types that captures this. That will make the model less valuable in these kinds of cases. If the Norwegian Police chooses to work further on a NER model to be used in investigations they should look into if there are any additional entity types that would be valuable for them.

Another issue with the dataset we created is that it is only annotated by one untrained annotator. Ideally the dataset should have been annotated by two annotators. There should in addition have been created a thorough guideline on how the data should be annotated. It should consist of guidelines on what each entity type should include and how to differentiate between for instance *location* and *organizations* for schools. This would have made the annotations more concise, which again would create a better model with better performance.

Some of the entity types were difficult to annotate, like *personal property*, *location* and *organization*, which affected the performance of these entity types. *Personal property*, as discussed in Section 5, Section 6.2 and Section 6.4 could be a lot of different things and it was often difficult to decide if something were a *personal property* or not. It was also often difficult to distinguish between *location* and *organization* for some tokens. This was especially difficult for schools and countries. We decided to not use the *geo-political* entity type, but only *location* and *organization*, and this was one of the causes for this difficulty. The difficulty of annotating these types lead to an inconsistency in the annotations in the dataset, which again lead to poorer performance. Having a thorough guideline for annotations and using two annotators would solve at least part of this problem.

6.4 Limitations

The NER model we have trained is trained on Wikipedia articles related to crime. Even though we get good performance when we test our model, it would probably not work as well in an actual investigation and with actual forensic data. This is because our data is not domain specific enough, there is a lot of difference between real investigative data and Wikipedia articles. Wikipedia articles gives structured summaries of cases, while real investigative data will be unstructured data from different sources. Our model could still bring value to an investigation and could be able to detect a lot of named entities, but a model that is trained on real forensic data will probably perform a lot better. If the Norwegian Police want to use a NER model in their work, they should therefore train their own model on real investigative data. We have shown that our approach for fine-tuning a Norwegian NER model works very well and achieves very good performance. The Norwegian Police could therefore use the same approach, just with their own data.

If the Norwegian Police created their own model with real forensic data the problem with the distribution of different crime types discussed in Section 6.3 would be solved. When using their own data the police would get a model that is trained on a representative distribution of case types, since it is actual cases the police have worked with. The distribution of data in the dataset will therefore be close to the real world. The same goes for the occurrences of named entities in the dataset. The data will be from actual cases which will have many occurrences of the domain specific entity types.

Even the best NER models are not able to get a perfect performance and recognize every named entity. In an investigation this can be crucial. If the investigators rely fully on the NER model and the model does not recognize everything, important evidence might not be found. It is therefore important to have a human in the loop, to make sure that important evidence is not overlooked. Machine learning models, like NER, could be a great tool in investigations, but it could never be 100% relied on. It is always possible that something important is missed. A human should therefore make sure that important evidence is not missed. Fabien *et al.* [47] also reports on the importance of this.

The NER model could be a great tool, but as it can not be relied a 100% on, it might not identify everything it should and it might classify tokens incorrect. Just looking at the identified entities without looking at the context might also be harmful. It might lead the investigators to people that does not actually have anything to do with the case and distract them from the actual important parts of the evidence. Parts of texts that does not contain any named entities, and words and phrases that are named entities, might also be important in an investigation. Relying too much on a NER model will then result in important evidence not being found.

The goal of a NER model is that it should be able to recognize new and unseen words. Our model recognizes a lot of new and unseen words and phrases, but it is not able to regonize everything. When it comes to the general entity types, they are often easier to identify since the entity type can be recognized by semantic, syntax or grammar. There is also more occurrences of these entity types in the dataset, which also makes the model better at recognizing them.

With the domain specific entity types on the other hand there is not as much characteristics to recognize them by. The context might help to recognize them, but the way the words are written are very similar for a car and a drug. Neither have capital letters in the start or anything other that identify the entity type. The same goes for the rest of the domain specific entity types. The *personal property* entity also could include a lot of these types, both a car and a weapon is most often a personal property, but we want the model to recognize that these specific personal properties belongs to a different entity type. Some of the domain specific entity types only includes a limited number of different words. There are a limited amount of different car brands, drug types, weapons and relations between people. With enough training data, the model will then still be able to identify the named entities because it has seen the words before. It might also be able to capture variations of the words and preceding and following words in a phrase.

For the *personal property* entity type the goal is that the model is able to identify personal properties based on the context. In addition it should be trained on enough data, so that it is able to identify if the words belong to any of the other domain specific entity types that are also personal properties or if it belong to the general *personal property* entity type.

Based on the performance of the model it looks like the model especially needs to be trained on more occurrences of *narcotic drug*, but also on *weapon*. It also looks like the model still has not learned to identify and differentiate on *personal property*. The recall is not so bad, so it recognizes most of the *personal property* entities, but the precision is only 0.605. This means that it classifies a lot more tokens as *personal properties* than it should. These tokens could both be other domain specific entities or not a named entity at all. One reason for this could be the difficulty of annotating this entity type. It could be that some *personal properties* are not classified correctly in the dataset and that the model here performs better than the human annotator. If this is not the case, the model needs to be trained on more occurrences of *personal properties*.

There is a great disadvantage with entity types that are recognized mostly based on already seen occurrences and not on context, syntax, semantic and grammar. New *drugs* emerge from time to time, and the NER model might then not be able to recognize them and identity them as the correct entity type. The same goes for *vehicles* and *weapons*. To be able to recognize new types of *drugs* and *weapons* we need a model that can recognize these entities only based on the context. Hopefully, a model that are trained on a large amount of real investigative data could be better at recognizing domain specific entity types based on context.

Another factor that might make it difficult to get good performance in an investigative setting and make it hard to recognize and identify all named entities is that texts from evidence might include dialects, slang, typos, words in other languages and different ways of writing. These words might be harder to identify and a lot of unseen words might not be recognized as named entities. Our model would probably not perform very well on dialects and slang, even though it might recognize some entities based on context. If the model were trained on real data the other hand, it would have been trained on dialects and slang and would therefore also be able to recognize these in unseen data. This shows the importance of training the model on the same kind of text that the model is going to be used on. The model might still encounter new slang words from time to time that it is not able to recognize.

We saw during the process of annotating the data, that the nb-bert-base-ner model doing the initial annotation for us, performed very poorly on unusual and foreign *names*. This is probably because there are few of these names in the training data so the model is not able to recognize them. In our training data there are more foreign and unusual names and our models therefore performs very well on *people names*. Training the model on real data instead of Wikipedia data will probably make the model even better at recognizing foreign and unusual *names* since it then is trained on a large diversity of names from real cases.

One way to deal with the issue of new kinds of *drugs*, new slang words and unusual *names* not being recognized is continuous training. This means that the model is trained more than once. The models is initially trained on a lot of data. When new data is then collected the model could be fine-tuned on this data as well. If new drugs, slang or dialects are then encountered, the model could be trained on this data, and it will be better at recognizing this type of text next time it encounters it. This is a great advantage, but it also requires additional resources. In addition to the human in the loop that makes sure that the model does not miss anything important, someone now needs to annotate all the new data and train a new model. There is also a risk of overfitting the model, so it also has to be checked that the performance of the models is not reduced. Training the model and checking the performance could be automated, but a human still needs to annotate the data.

6.4.1 Legal and Ethical Considerations

Before training their own model, the Norwegian Police needs to look into the legal aspect of using real investigation data to train a NER model and make sure that everything is done according to current laws and regulations. Investigation data contains a lot of personal and sensitive information and it needs to be checked out if it is legal to train a ML model with this data. There might be laws and rules about how long the police is allowed to store data from closed cases and what they can use it for. It therefore needs to be checked if it is legal to train a NER model with this data. After a model is trained there is no way to deduce the data the model is trained on, so it is a possibility to train the model and then delete the training data. In that case, when training new models, previous models should be saved in case some of the new data that is used for training decreases the performance and it is needed to go back to a previous version of the model.

The police also have to be careful about how they use the model. ML models could be biased and it is therefore important to be careful when using it. It should be used in an ethical and fair way. Zardiashvili *et al.* [48] discusses the ethics of using AI in the Dutch police. They find 6 requirements for using AI in an ethical way:

Accountability Someone needs to be held accountable for what the AI system does.

Transparency How much transparency, and about what and to whom, have to be discussed and planned.

Privacy and Data Protection The privacy of people needs to be considered when using AI in police work.

Fairness and Inclusivity It has to be made sure the the AI systems works in a fair way. A biased system might lead to discriminatory treatment towards groups in the population.

Human Autonomy and Agency An AI system is not perfect and humans should still be able to make decisions freely.

(Socio-technical) Robustness and Safety When developing AI systems that are going to be used in the police the risks and benefits that it comes with needs to be taken into consideration and the systems needs to be robust and safe to use.

The Europeian Union (EU) have proposed a law on AI and the use of AI, the AI Act, which are especially compliant for high risk systems. The police have to make sure that their use is compliant with this law [49].

6.5 Nynorsk and Other Languages

Real investigation data could contain a mix of languages and our model would not be able to recognize named entities in other languages. One way to deal with this model is to create a multilingual model that is trained on multiple languages, but this could demand a lot more resources. As already mentioned, the police should use real investigation data if they if choose to create a NER model. Using real investigation data will also help with this problem, since the data the model is trained on then contains a mix of different languages, and also the languages the police most often encounters. The same also goes for the other written language of Norwegian, Nynorsk.

Chapter 7

Conclusion

The police in general, and more specifically the Norwegian Police, is facing a challenge with the digital forensics backlog and the large amount of digital devices that needs to be examined in investigations. All the text that these devices contain is time consuming to read through, and the police could benefit from a tool that could help them locate important content. Our research have created a proof of concept for a Norwegian NER model for investigative purposes. Using Wikipedia data we were able to create a domain specific NER model by fine-tuning a BERT model. We fine-tuned the model on Wikipedia data we annotated with general and domain specific entity types, like person, drugs, weapons and relations. After doing experiments with different amounts of training data and different hyper-paramteres we ended up with our optimized model. This model achieved good results and performs as well as both other NER models created for investigative purposes for other languages and the state-of-the-art general Norwegian NER models. Our model achieved a precision of 0.904, a recall of 0.908 and a F1-score of 0.906. The entity type with the highest score was *vehicle*, with a F1-score of 0.973. Of the domain specific entity types *weapon* and *relation* also had good performance with F1-scores of 0.851 and 0.930.

Our model is still not perfect, but a good proof of concept. It shows that the police could create their own NER model for use in investigations and that they could achieve fairly good results with not too much data. To get a model that performs well on every entity type, they need to train their model on a sufficient amount of occurrences of each entity type, but using real investigation data for training could solve this problem. No matter how good performance they are able to achieve, the model will not be perfect, and in an investigation there is a lot at stake. It is therefore important to be careful when using the model and always have a human in the loop.

7.1 Future Work

The investigator we talked to said that it might be difficult to create a NER model that could detect all interesting entities and that detecting weapons and drugs is not always relevant. Humans on the other hand is always interesting. In addition the relations humans have to objects, places, organizations and other people could be very relevant and interesting in investigations. It could therefore be valuable to look into creating a information extraction system for Norwegian investigations in the future.

Both Rodrigues *et al.* [39] and Yang and Chow [7] created information extraction system to use in investigation. They both created systems that were going to help investigators by locating interesting entities and also finding relations between the entities. Both systems have a NER system that first extract named entities before different relation extraction approaches were used to find the relations. Yang and Chow [7] uses a classic Apriori algorithm to extract relations while Rodrigues *et al.* [39] uses supervised learning models. Both methods finds relations between the entities, just in different ways. In future research on creating an information extraction system for investigations in Norwegian, it should be looked into which method is more favorable to use.

Extracting relations in addition to just the named entities could be of great value in an investigation and further help the investigators to find interesting evidence and clues. If the investigators finds an entity of interest they can use the extracted relations to figure out where to look for further evidence and valuable information. This could help them save a lot of time.

Another thing that could be interesting to look at in the future, which the investigator we talked to pointed out, is to create a system that could take input data to help the model find interesting and relevant data. In an investigation where there is already some central objects, locations or persons, it would be interesting to see how a system could use that as input data to get different results. The system could then extract information that are related to the input data, which the investigators already know is important. Such a system could help the investigators quicker locate important evidence compared to a regular NER system or an information extraction system, but it requires that some entities are already known to be of importance. An optimal solution could be to create a system that works both with and without input data.

When talking to the investigator we were reminded that the cases the police investigate are very different. There are different things that are important in a murder case and in a fraud case. It is therefore difficult to create a general NER model that works well and brings value in all investigations. A solution could be to create different models for different types of cases. Each model could then be trained on data from just the types of cases it is going to be used on and this will probably increase the performance of the models and make them good at detecting interesting entities in all types of cases. On the other hand, this would lead to more work when creating and training the models.

Bibliography

[1] A. Årnes, *Digital Forensics*. Wiley, 2018.

[2] M. Scanlon, 'Battling the digital forensic backlog through data deduplication,' Aug. 2016. DOI: `10.1109/INTECH.2016.7845139`.

[3] E. Casey, M. Ferraro and L. Nguyen, 'Investigation delayed is justice denied: Proposals for expediting forensic examinations of digital evidence,' *Journal of forensic sciences*, vol. 54, pp. 1353–64, Sep. 2009. DOI: `10.1111/j.1556-4029.2009.01150.x`.

[4] X. Du, C. Hargreaves, J. Sheppard, F. Anda, A. Sayakkara, N.-A. Le-Khac and M. Scanlon, 'SoK: Exploring the State of the Art and the Future Potential of Artificial Intelligence in Digital Forensic Investigation,' in *The 13th International Workshop on Digital Forensics (WSDF), held at the 15th International Conference on Availability, Reliability and Security (ARES)*, ser. ARES '20, Dublin, Ireland: ACM, Aug. 2020.

[5] D. O. Ukwen and M. Karabatak, 'Review of nlp-based systems in digital forensics and cybersecurity,' in *2021 9th International Symposium on Digital Forensics and Security (ISDFS)*, 2021, pp. 1–9. DOI: `10.1109/ISDFS52919.2021.9486354`.

[6] B. Mohit, 'Named entity recognition,' in *Natural Language Processing of Semitic Languages*, I. Zitouni, Ed., Springer, USA, 2014, ISBN: 978-3642453571.

[7] M. Yang and K.-P. Chow, 'An information extraction framework for digital forensic investigations,' in *Advances in Digital Forensics XI*, G. Peterson and S. Shenoi, Eds., Cham: Springer International Publishing, 2015, pp. 61–76, ISBN: 978-3-319-24123-4.

[8] M. Chau, J. J. Xu and H. Chen, 'Extracting meaningful entities from police narrative reports,' in *Proceedings of the 2002 Annual National Conference on Digital Government Research*, Digital Government Society of North America, 2002.

[9] W. Wu, K.-P. Chow, Y. Mai and J. Zhang, 'Public opinion monitoring for proactive crime detection using named entity recognition,' in *Advances in Digital Forensics XVI*, G. Peterson and S. Shenoi, Eds., Cham: Springer International Publishing, 2020, pp. 203–214, ISBN: 978-3-030-56223-6.

[10] F. Jørgensen, T. Aasmoe, A.-S. Ruud Husevåg, L. Øvrelid and E. Velldal, 'NorNE: Annotating named entities for Norwegian,' English, in *Proceedings of the Twelfth Language Resources and Evaluation Conference*, Marseille, France: European Language Resources Association, May 2020, pp. 4547–4556, ISBN: 979-10-95546-34-4. [Online]. Available: `https://aclanthology.org/2020.lrec-1.559`.

[11] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd edition. Pearson, 2009, pp. 727–737, ISBN: 978-0136042594.

[12] MathWorks. 'What is deep learning? 3 things you need to know.' (), [Online]. Available: `https://se.mathworks.com/discovery/deep-learning.html` (visited on 01/06/2023).

[13] P. M. Nadkarni, L. Ohno-Machado and W. W. Chapman, 'Natural language processing: an introduction,' *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, Sep. 2011, ISSN: 1067-5027. DOI: `10.1136/amiajnl-2011-000464`. eprint: `https://academic.oup.com/jamia/article-pdf/18/5/544/5962687/18-5-544.pdf`. [Online]. Available: `https://doi.org/10.1136/amiajnl-2011-000464`.

[14] D. Nadeau and S. Sekine, 'A survey of named entity recognition and classification,' *Lingvisticae Investigationes*, vol. 30, Aug. 2007. DOI: `10.1075/li.30.1.03nad`.

[15] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami and C. Dyer, *Neural architectures for named entity recognition*, 2016. arXiv: `1603.01360 [cs.CL]`.

[16] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest and A. Rush, 'Transformers: State-of-the-art natural language processing,' in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. DOI: `10.18653/v1/2020.emnlp-demos.6`. [Online]. Available: `https://aclanthology.org/2020.emnlp-demos.6`.

[17] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2018. DOI: `10.48550/ARXIV.1810.04805`. [Online]. Available: `https://arxiv.org/abs/1810.04805`.

[18] N. Reimers and I. Gurevych, *Optimal hyperparameters for deep lstm-networks for sequence labeling tasks*, 2017. arXiv: `1707.06799 [cs.CL]`.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, 'Attention is all you need,' in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[20] Y. Jia, 'Attention mechanism in machine translation,' *Journal of Physics: Conference Series*, vol. 1314, p. 012 186, Oct. 2019. DOI: `10.1088/1742-6596/1314/1/012186`.

[21] M. Kapronczay. 'A beginner's guide to language models.' (), [Online]. Available: `https://builtin.com/data-science/beginners-guide-language-models` (visited on 08/06/2023).

[22] Scaler. 'Masked language modeling in bert.' (), [Online]. Available: `https://www.scaler.com/topics/nlp/masked-language-model-explained/` (visited on 12/06/2023).

[23] Y. Zhu, R. Kiros, R. S. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba and S. Fidler, 'Aligning books and movies: Towards story-like visual explanations by watching movies and reading books,' *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 19–27, 2015.

[24] P. E. Kummervold, J. De la Rosa, F. Wetjen and S. A. Brygfjeld, 'Operationalizing a national digital library: The case for a Norwegian transformer model,' in *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*, Reykjavik, Iceland (Online): Linköping University Electronic Press, Sweden, 2021, pp. 20–29. [Online]. Available: `https://aclanthology.org/2021.nodalida-main.3`.

[25] A. Kutuzov, J. Barnes, E. Velldal, L. Øvrelid and S. Oepen, 'Large-scale contextualised language modelling for Norwegian,' in *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*, Reykjavik, Iceland (Online): Linköping University Electronic Press, Sweden, May 2021, pp. 30–40. [Online]. Available: `https://aclanthology.org/2021.nodalida-main.4`.

[26] P. E. Solberg, A. Skjærholt, L. Øvrelid, K. Hagen and J. B. Johannessen, 'The Norwegian dependency treebank,' in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland: European Language Resources Association (ELRA), May 2014, pp. 789–795. [Online]. Available: `http://www.lrec-conf.org/proceedings/lrec2014/pdf/303_Paper.pdf`.

[27] K. P. Shung. 'Accuracy, precision, recall or f1?' (2018), [Online]. Available: `https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9` (visited on 31/05/2023).

[28] S. L. Garfinkel, A. Parker-Wood, D. Huynh and J. Migletz, 'An automated solution to the multiuser carved data ascription problem,' *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 4, pp. 868–882, 2010. DOI: 10.1109/TIFS.2010.2060484.

[29] F. Marturana, G. Me, R. Berte and S. Tacconi, 'A quantitative approach to triaging in mobile forensics,' in *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, 2011, pp. 582–588. DOI: 10.1109/TrustCom.2011.75.

[30] M. Khan, C. Chatwin and R. Young, 'A framework for post-event timeline reconstruction using neural networks,' *Digital Investigation*, vol. 4, no. 3, pp. 146–157, 2007, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2007.11.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287607000837.

[31] S. Fitzgerald, G. Mathews, C. Morris and O. Zhulyn, 'Using nlp techniques for file fragment classification,' *Digital Investigation*, vol. 9, S44–S49, 2012, The Proceedings of the Twelfth Annual DFRWS Conference, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2012.05.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287612000333.

[32] L. Du, H. Jin, O. de Vel and N. Liu, 'A latent semantic indexing and wordnet based information retrieval model for digital forensics,' in *2008 IEEE International Conference on Intelligence and Security Informatics*, 2008, pp. 70–75. DOI: 10.1109/ISI.2008.4565032.

[33] A. de Waal, J. Venter and E. Barnard, 'Applying topic modeling to forensic data,' vol. 285, Jan. 2008, ISBN: 978-0-387-84926-3. DOI: 10.1007/978-0-387-84927-0_10.

[34] D. R. O'Day and R. A. Calix, 'Text message corpus: Applying natural language processing to mobile device forensics,' in *2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, 2013, pp. 1–6. DOI: 10.1109/ICMEW.2013.6618380.

[35] S. Keretna, A. Hossny and D. Creighton, 'Recognising user identity in twitter social networks via text mining,' in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, 2013, pp. 3079–3082. DOI: 10.1109/SMC.2013.525.

[36] R. Y. Lau, Y. Xia and Y. Ye, 'A probabilistic generative model for mining cybercriminal networks from online social media,' *IEEE Computational Intelligence Magazine*, vol. 9, no. 1, pp. 31–43, 2014. DOI: 10.1109/MCI.2013.2291689.

[37] D. Sun, X. Zhang, K.-K. R. Choo, L. Hu and F. Wang, 'Nlp-based digital forensic investigation platform for online communications,' *Computers Security*, vol. 104, p. 102 210, Jan. 2021. DOI: 10.1016/j.cose.2021.102210.

[38] B. Klimt and Y. Yang, 'Introducing the enron corpus,' in *International Conference on Email and Anti-Spam*, 2004.

[39] F. B. Rodrigues, W. F. Giozza, R. de Oliveira Albuquerque and L. J. García Villalba, 'Natural language processing applied to forensics information extraction with transformers and graph visualization,' *IEEE Transactions on Computational Social Systems*, pp. 1–17, 2022. DOI: 10.1109/TCSS.2022.3159677.

[40] H. Shabat, N. Omar and K. Rahem, 'Named entity recognition in crime using machine learning approach,' Dec. 2014, pp. 280–288, ISBN: 978-3-319-12843-6. DOI: 10.1007/978-3-319-12844-3_24.

[41] P. Kummervold, F. Wetjen and J. de la Rosa, 'The Norwegian colossal corpus: A text corpus for training large Norwegian language models,' in *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, Marseille, France: European Language Resources Association, Jun. 2022, pp. 3852–3860. [Online]. Available: https://aclanthology.org/2022.lrec-1.410.

[42] B. Johansen, 'Named-entity recognition for Norwegian,' in *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, Turku, Finland: Linköping University Electronic Press, Sep. 2019, pp. 222–231. [Online]. Available: https://aclanthology.org/W19-6123.

[43] J. Johannessen, K. Hagen, Å. Haaland, A. Jónsdottir, A. Nøklestad, D. Kokkinakis, P. Meurer, E. Bick and D. Hansen, 'Named entity recognition for the mainland scandinavian languages,' *Literary and Linguistic Computing*, vol. 20, Feb. 2005. DOI: 10.1093/llc/fqh045.

[44] C.-H. Ku, A. Iriberri and G. Leroy, 'Crime information extraction from police and witness narrative reports,' Jun. 2008, pp. 193–198, ISBN: 978-1-4244-1977-7. DOI: 10.1109/THS.2008.4534448.

[45] H. Nakayama, *seqeval: A python framework for sequence labeling evaluation*, Software available from https://github.com/chakki-works/seqeval, 2018. [Online]. Available: https://github.com/chakki-works/seqeval.

[46] D. Lillis and M. Scanlon, 'On the benefits of information retrieval and information extraction techniques applied to digital forensics,' in Jan. 2016, vol. 393, pp. 641–647, ISBN: 978-981-10-1535-9. DOI: 10.1007/978-981-10-1536-6_83.

[47] M. Fabien, S. Parida, P. Motlicek, D. Zhu, A. Krishnan and H. H. Nguyen, 'ROXANNE Research Platform: Automate Criminal Investigations,' in *Proc. Interspeech 2021*, 2021, pp. 962–964.

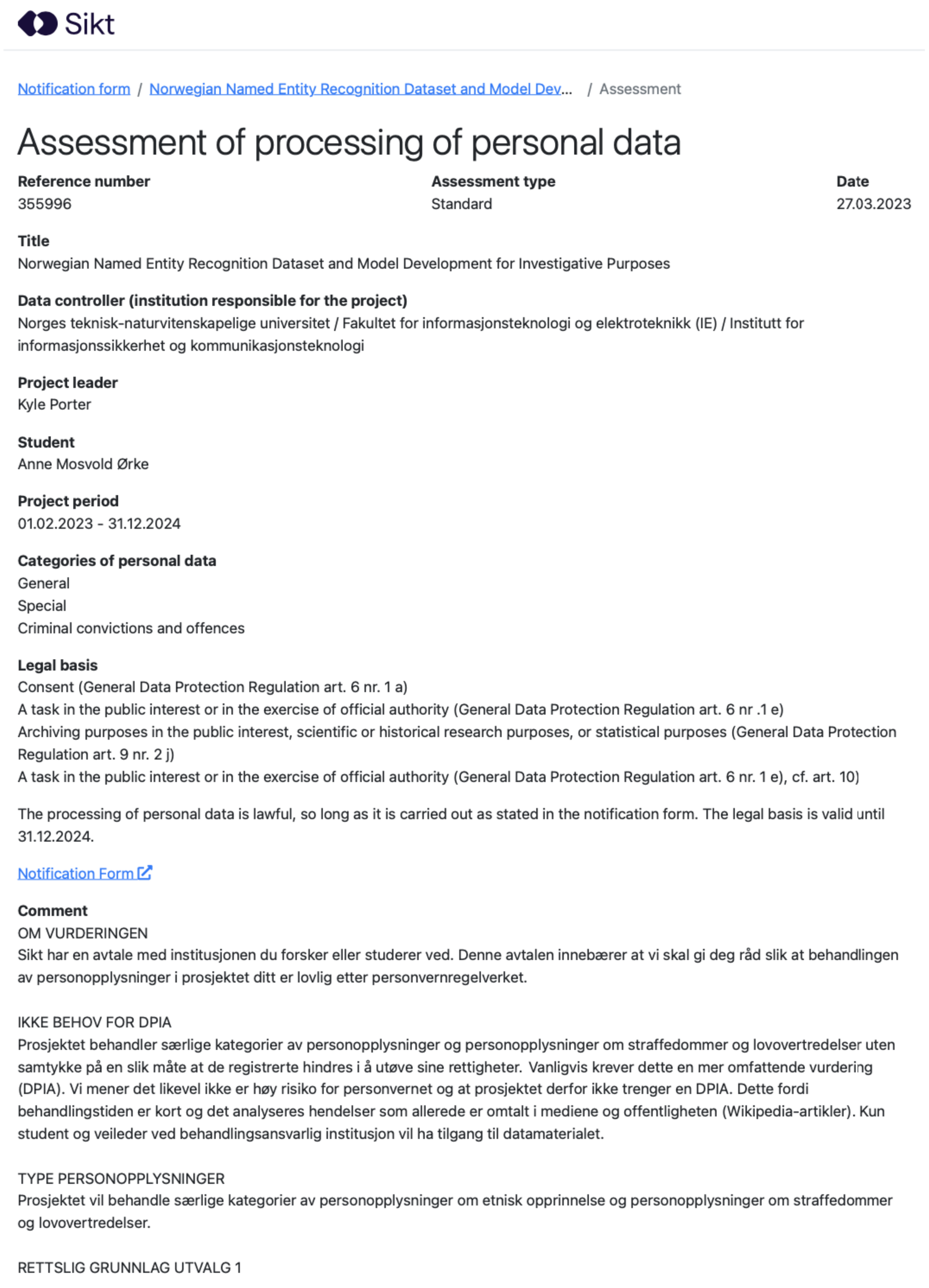
[48] L. Zardiashvili, J. Bieger, F. Dechesne and V. Dignum, 'Ai ethics for law enforcement: A study into requirements for responsible use of ai at the dutch police,' *Delphi - Interdisciplinary Review of Emerging Technologies*, vol. 2, pp. 179–185, 4 2019. DOI: 10.21552/delphi/2019/4/7.

[49] T. A. Act. 'The artificial intelligence act.' (), [Online]. Available: `https://artificialintelligenceact.eu` (visited on 12/07/2023).

Appendix A

Sikt Application Assessment

The assessment of our Sikt application:



Sikt

Notification form / Norwegian Named Entity Recognition Dataset and Model Dev... / Assessment

Assessment of processing of personal data

Reference number	**Assessment type**	**Date**
355996	Standard	27.03.2023

Title
Norwegian Named Entity Recognition Dataset and Model Development for Investigative Purposes

Data controller (institution responsible for the project)
Norges teknisk-naturvitenskapelige universitet / Fakultet for informasjonsteknologi og elektroteknikk (IE) / Institutt for informasjonssikkerhet og kommunikasjonsteknologi

Project leader
Kyle Porter

Student
Anne Mosvold Ørke

Project period
01.02.2023 - 31.12.2024

Categories of personal data
General
Special
Criminal convictions and offences

Legal basis
Consent (General Data Protection Regulation art. 6 nr. 1 a)
A task in the public interest or in the exercise of official authority (General Data Protection Regulation art. 6 nr .1 e)
Archiving purposes in the public interest, scientific or historical research purposes, or statistical purposes (General Data Protection Regulation art. 9 nr. 2 j)
A task in the public interest or in the exercise of official authority (General Data Protection Regulation art. 6 nr. 1 e), cf. art. 10)

The processing of personal data is lawful, so long as it is carried out as stated in the notification form. The legal basis is valid until 31.12.2024.

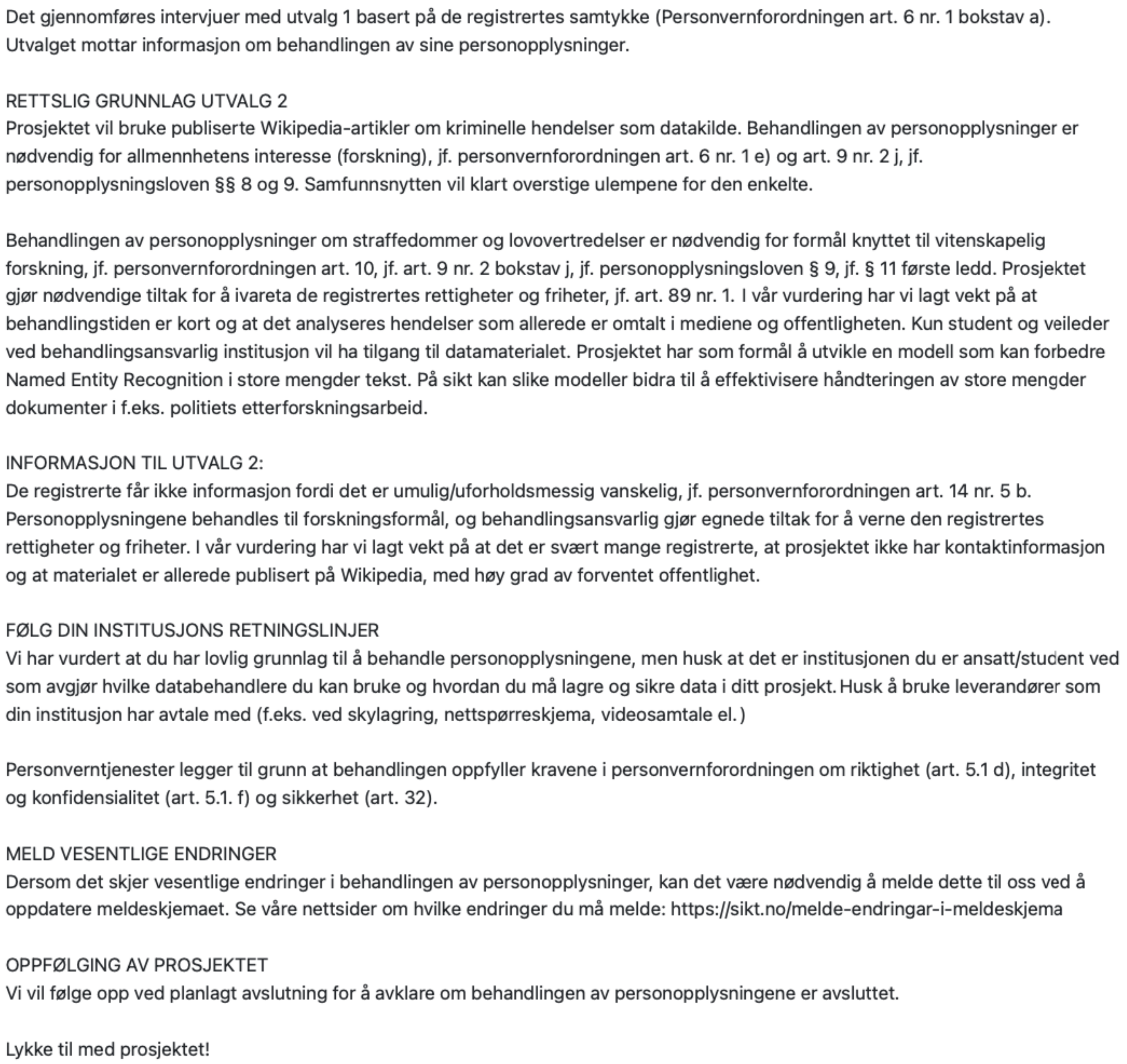
Notification Form

Comment
OM VURDERINGEN
Sikt har en avtale med institusjonen du forsker eller studerer ved. Denne avtalen innebærer at vi skal gi deg råd slik at behandlingen av personopplysninger i prosjektet ditt er lovlig etter personvernregelverket.

IKKE BEHOV FOR DPIA
Prosjektet behandler særlige kategorier av personopplysninger og personopplysninger om straffedommer og lovovertredelser uten samtykke på en slik måte at de registrerte hindres i å utøve sine rettigheter. Vanligvis krever dette en mer omfattende vurdering (DPIA). Vi mener det likevel ikke er høy risiko for personvernet og at prosjektet derfor ikke trenger en DPIA. Dette fordi behandlingstiden er kort og det analyseres hendelser som allerede er omtalt i mediene og offentligheten (Wikipedia-artikler). Kun student og veileder ved behandlingsansvarlig institusjon vil ha tilgang til datamaterialet.

TYPE PERSONOPPLYSNINGER
Prosjektet vil behandle særlige kategorier av personopplysninger om etnisk opprinnelse og personopplysninger om straffedommer og lovovertredelser.

RETTSLIG GRUNNLAG UTVALG 1



Det gjennomføres intervjuer med utvalg 1 basert på de registrertes samtykke (Personvernforordningen art. 6 nr. 1 bokstav a). Utvalget mottar informasjon om behandlingen av sine personopplysninger.

RETTSLIG GRUNNLAG UTVALG 2
Prosjektet vil bruke publiserte Wikipedia-artikler om kriminelle hendelser som datakilde. Behandlingen av personopplysninger er nødvendig for allmennhetens interesse (forskning), jf. personvernforordningen art. 6 nr. 1 e) og art. 9 nr. 2 j, jf. personopplysningsloven §§ 8 og 9. Samfunnsnytten vil klart overstige ulempene for den enkelte.

Behandlingen av personopplysninger om straffedommer og lovovertredelser er nødvendig for formål knyttet til vitenskapelig forskning, jf. personvernforordningen art. 10, jf. art. 9 nr. 2 bokstav j, jf. personopplysningsloven § 9, jf. § 11 første ledd. Prosjektet gjør nødvendige tiltak for å ivareta de registrertes rettigheter og friheter, jf. art. 89 nr. 1. I vår vurdering har vi lagt vekt på at behandlingstiden er kort og at det analyseres hendelser som allerede er omtalt i mediene og offentligheten. Kun student og veileder ved behandlingsansvarlig institusjon vil ha tilgang til datamaterialet. Prosjektet har som formål å utvikle en modell som kan forbedre Named Entity Recognition i store mengder tekst. På sikt kan slike modeller bidra til å effektivisere håndteringen av store mengder dokumenter i f.eks. politiets etterforskningsarbeid.

INFORMASJON TIL UTVALG 2:
De registrerte får ikke informasjon fordi det er umulig/uforholdsmessig vanskelig, jf. personvernforordningen art. 14 nr. 5 b. Personopplysningene behandles til forskningsformål, og behandlingsansvarlig gjør egnede tiltak for å verne den registrertes rettigheter og friheter. I vår vurdering har vi lagt vekt på at det er svært mange registrerte, at prosjektet ikke har kontaktinformasjon og at materialet er allerede publisert på Wikipedia, med høy grad av forventet offentlighet.

FØLG DIN INSTITUSJONS RETNINGSLINJER
Vi har vurdert at du har lovlig grunnlag til å behandle personopplysningene, men husk at det er institusjonen du er ansatt/student ved som avgjør hvilke databehandlere du kan bruke og hvordan du må lagre og sikre data i ditt prosjekt. Husk å bruke leverandører som din institusjon har avtale med (f.eks. ved skylagring, nettspørreskjema, videosamtale el.)

Personverntjenester legger til grunn at behandlingen oppfyller kravene i personvernforordningen om riktighet (art. 5.1 d), integritet og konfidensialitet (art. 5.1. f) og sikkerhet (art. 32).

MELD VESENTLIGE ENDRINGER
Dersom det skjer vesentlige endringer i behandlingen av personopplysninger, kan det være nødvendig å melde dette til oss ved å oppdatere meldeskjemaet. Se våre nettsider om hvilke endringer du må melde: https://sikt.no/melde-endringar-i-meldeskjema

OPPFØLGING AV PROSJEKTET
Vi vil følge opp ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet.

Lykke til med prosjektet!

Appendix B

Scraping Script

Code listing B.1: Script to collect and structure Wikipedia files

```
from bs4 import BeautifulSoup
import requests
import csv
import re

article_file = open('articles.txt', 'r')

articles = article_file.read()

article_list = articles.split('\n')

with open('articles.csv', 'w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["Title", "Text"])

for article in article_list:
        if not article == '':
                page = requests.get(article)
                soup = BeautifulSoup(page.content, 'lxml')
                p_list = soup.find_all('p')
                full_text = ''
                for p in p_list:
                        p_text = p.get_text().replace('\n', '')
                        p_text = re.sub('\[[0-9]+\]', '', p_text)
                        full_text += '␣' + p_text
                with open('articles.csv', 'a', newline='') as file:
                        writer = csv.writer(file)
                        writer.writerow([soup.find('h1').get_text(), full_text])
```

Appendix C

Annotation Script

Code listing C.1: Script to do initial annotation and structure data in json files

```
#https://huggingface.co/NbAiLab/nb-bert-base-ner

from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline
import csv
import re
import json

tokenizer = AutoTokenizer.from_pretrained("NbAiLab/nb-bert-base-ner",
                 model_max_length=512)
model = AutoModelForTokenClassification.from_pretrained("NbAiLab/nb-bert-base-ner")

ner = pipeline("ner", model=model, tokenizer=tokenizer)

with open("beautifulSoup/articles.csv", "r") as file:
   csvreader = csv.reader(file)
   next(csvreader)
   for row in csvreader:
      file = open('files/'+row[0]+'.json', 'w')
      file = open('files/'+row[0]+'.json', 'a')
      file.write('[')
      sentences = row[1].split('.')
      idx = 1
      for sentence in sentences:
         if len(sentence) <= 512:
            ner_result = ner(sentence)
            sentence_object = {}
            sentence_object["idx"] = str(idx)
            idx += 1
            sentence_object["lang"] = "bokmaal"
            sentence_object["text"] = sentence
            tokens = re.split('([^a-zA-Z0-9\'\-æøåÆ  Å])', sentence)
            tokens = [char for char in tokens if char != '␣']
            tokens = [char for char in tokens if char != '']
            sentence_object["tokens"] = tokens
            lemmas = [word.lower() for word in tokens]
            sentence_object["lemmas"] = lemmas
            ner_tags = []
            for word in tokens:
               ner_match = list(filter(lambda j: j['word'] == word, ner_result))
               if len(ner_match) > 0:
                  ner_tags.append(ner_match[0]['entity'])
               else:
                  ner_tags.append("O")
            sentence_object["ner_tags"] = ner_tags
            json_object = json.dumps(sentence_object, indent=6, ensure_ascii=False)
            file.write(json_object)
            file.write(',')
         else:
            print(sentence)
            print("Sentence␣too␣long")
      file = open('files/'+row[0]+'.json', 'rb+')
      file.seek(-1, 2)
      file.truncate()
      file = open('files/'+row[0]+'.json', 'a')
      file.write(']')
```

Appendix D

Split Script

Code listing D.1: Script to split the annotated data in train, validation and test files

```
import os
import json
import random
import numpy as np

files = os.listdir('files')
random.seed(42)
random.shuffle(files)

train_files, validation_files, test_files = np.split(files,
                       [int(len(files)*0.7), int(len(files)*0.9)])
overview_file = open('data/overview.txt', 'w')

def list_in_overview(files, title):
   overview_file.write('\n'+title+'\n')
   for file in files:
      overview_file.write(file+'\n')

def merge_files(files, file_name):
   all_sentences = []
   for file in files:
      file = open('files/'+file, 'r')
      all_sentences.extend(json.load(file))
   merged_file = open('data/'+file_name+'.json', 'w')
   json_objects = json.dumps(all_sentences, indent=6, ensure_ascii=False)
   merged_file.write(json_objects)


merge_files(train_files, 'train')
merge_files(validation_files, 'val')
merge_files(test_files, 'test')

list_in_overview(train_files, 'train')
list_in_overview(validation_files, 'val')
list_in_overview(test_files, 'test')
```

Appendix E

Training Script

Code listing E.1: Script to fine-tune and test the model

```
#!/usr/bin/env python
# coding=utf-8

# The original file is modified
# Copyright 2020 The HuggingFace Team All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
"""
Fine-tuning the library models for token classification.
"""
# You can also adapt this script on your own token classification task and datasets.
# Pointers for this are left as comments.

import logging
import os
import sys
from dataclasses import dataclass, field
from typing import Optional

import datasets
import evaluate
import numpy as np
from datasets import ClassLabel, load_dataset

import transformers
from transformers import (
    AutoConfig,
    AutoModelForTokenClassification,
    AutoTokenizer,
    DataCollatorForTokenClassification,
    HfArgumentParser,
    PretrainedConfig,
    PreTrainedTokenizerFast,
    Trainer,
    TrainingArguments,
    set_seed,
)
from transformers.trainer_utils import get_last_checkpoint
from transformers.utils import check_min_version, send_example_telemetry
from transformers.utils.versions import require_version


# Will error if the minimal version of
# Transformers is not installed. Remove at your own risks.
check_min_version("4.27.0.dev0")

require_version(
    "datasets>=1.8.0",
    "To␣fix:␣pip␣install␣-r␣examples/pytorch/token-classification/requirements.txt")

logger = logging.getLogger(__name__)


@dataclass
class ModelArguments:
    """
    Arguments pertaining to which model/config/tokenizer
    we are going to fine-tune from.
    """

    model_name_or_path: str = field(
        metadata={
            "help": (
            "Path␣to␣pretrained␣model␣or␣model␣identifier␣"
            "from␣huggingface.co/models")}
    )
    config_name: Optional[str] = field(
        default=None,
        metadata={
            "help": (
            "Pretrained␣config␣name␣or␣path␣if␣not␣the␣same␣"
            "as␣model_name")}
    )
    tokenizer_name: Optional[str] = field(
        default=None,
        metadata={
            "help": (
            "Pretrained␣tokenizer␣name␣or␣path␣if␣not␣the␣same␣"
            "as␣model_name")}
    )
    cache_dir: Optional[str] = field(
        default=None,
        metadata={
            "help": (
            "Where␣do␣you␣want␣to␣store␣the␣pretrained␣models␣"
            "downloaded␣from␣huggingface.co")},
    )
    model_revision: str = field(
        default="main",
        metadata={
            "help": (
            "The␣specific␣model␣version␣to␣use␣(can␣be␣a␣"
            "branch␣name,␣tag␣name␣or␣commit␣id).")},
    )
    use_auth_token: bool = field(
        default=False,
        metadata={
            "help": (
                "Will␣use␣the␣token␣generated␣when␣running␣"
                "‘huggingface-cli␣login‘␣(necessary␣to␣use␣this␣script␣"
                "with␣private␣models)."
            )
        },
    )
    ignore_mismatched_sizes: bool = field(
        default=False,
        metadata={
            "help": (
            "Will␣enable␣to␣load␣a␣pretrained␣model␣whose␣"
            "head␣dimensions␣are␣different.")},
    )


@dataclass
class DataTrainingArguments:
    """
    Arguments pertaining to what data we are going
    to input our model for training and eval.
    """

    task_name: Optional[str] = field(
        default="ner",
        metadata={"help": "The␣name␣of␣the␣task␣(ner,␣pos...)."})
    dataset_name: Optional[str] = field(
        default=None,
        metadata={
            "help":
            "The␣name␣of␣the␣dataset␣to␣use␣(via␣the␣datasets␣library)."}
    )
    dataset_config_name: Optional[str] = field(
        default=None,
        metadata={
            "help": (
            "The␣configuration␣name␣of␣the␣dataset␣"
            "to␣use␣(via␣the␣datasets␣library).")}
    )
    train_file: Optional[str] = field(
        default=None,
        metadata={
            "help":
            "The␣input␣training␣data␣file␣(a␣csv␣or␣JSON␣file)."}
    )
    validation_file: Optional[str] = field(
        default=None,
        metadata={
            "help": (
            "An␣optional␣input␣evaluation␣data␣file␣to␣"
            "evaluate␣on␣(a␣csv␣or␣JSON␣file).")},
    )
    test_file: Optional[str] = field(
        default=None,
        metadata={
            "help": (
            "An␣optional␣input␣test␣data␣file␣to␣"
            "predict␣on␣(a␣csv␣or␣JSON␣file).")},
    )
    text_column_name: Optional[str] = field(
        default=None,
        metadata={
            "help": (
            "The␣column␣name␣of␣text␣to␣input␣in␣the␣"
            "file␣(a␣csv␣or␣JSON␣file).")}
    )
    label_column_name: Optional[str] = field(
        default=None,
        metadata={
            "help": (
            "The␣column␣name␣of␣label␣to␣input␣in␣the␣"
            "file␣(a␣csv␣or␣JSON␣file).")}
    )
    overwrite_cache: bool = field(
        default=False,
        metadata={
            "help":
            "Overwrite␣the␣cached␣training␣and␣evaluation␣sets"}
    )
    preprocessing_num_workers: Optional[int] = field(
        default=None,
        metadata={
            "help":
            "The␣number␣of␣processes␣to␣use␣for␣the␣preprocessing."},
    )
    max_seq_length: int = field(
        default=None,
        metadata={
            "help": (
                "The␣maximum␣total␣input␣sequence␣length␣after␣"
                "tokenization.␣If␣set,␣sequences␣longer␣than␣"
                "this␣will␣be␣truncated,␣sequences␣shorter␣will␣be␣padded."
            )
        },
    )
    pad_to_max_length: bool = field(
        default=False,
        metadata={
            "help": (
                "Whether␣to␣pad␣all␣samples␣to␣model␣maximum␣sentence␣length.␣"
                "If␣False,␣will␣pad␣the␣samples␣dynamically␣"
                "when␣batching␣to␣the␣maximum␣length␣in␣the␣batch.␣More␣"
                "efficient␣on␣GPU␣but␣very␣bad␣for␣TPU."
            )
        },
    )
    max_train_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "For␣debugging␣purposes␣or␣quicker␣training,␣"
                "truncate␣the␣number␣of␣training␣examples␣to␣this␣"
                "value␣if␣set."
            )
        },
    )
    max_eval_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "For␣debugging␣purposes␣or␣quicker␣training,␣"
                "truncate␣the␣number␣of␣evaluation␣examples␣to␣this␣"
                "value␣if␣set."
            )
        },
    )
    max_predict_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "For␣debugging␣purposes␣or␣quicker␣training,␣"
                "truncate␣the␣number␣of␣prediction␣examples␣to␣this␣"
                "value␣if␣set."
            )
        },
    )
    label_all_tokens: bool = field(
        default=False,
        metadata={
            "help": (
                "Whether␣to␣put␣the␣label␣for␣one␣word␣on␣all␣"
                "tokens␣of␣generated␣by␣that␣word␣or␣just␣on␣the␣"
                "one␣(in␣which␣case␣the␣other␣tokens␣will␣"
                "have␣a␣padding␣index)."
            )
        },
    )
    return_entity_level_metrics: bool = field(
        default=False,
        metadata={
            "help": (
            "Whether␣to␣return␣all␣the␣entity␣levels␣during␣"
            "evaluation␣or␣just␣the␣overall␣ones.")},
    )

    def __post_init__(self):
        if (self.dataset_name is None and self.train_file is None
        and self.validation_file is None):
            raise ValueError("Need␣either␣a␣dataset␣name␣or␣a␣"
                             "training/validation␣file.")
        else:
            if self.train_file is not None:
                extension = self.train_file.split(".")[-1]
                assert extension in ["csv", "json"], ("‘train_file‘␣should␣"
                                                      "be␣a␣csv␣or␣a␣json␣file.")
            if self.validation_file is not None:
                extension = self.validation_file.split(".")[-1]
                assert extension in ["csv", "json"], ("‘validation_file‘␣should␣"
                                                      "be␣a␣csv␣or␣a␣json␣file.")
        self.task_name = self.task_name.lower()


def main():
```

```
# See all possible arguments in src/transformers/training_args.py
# or by passing the --help flag to this script.
# We now keep distinct sets of args, for a cleaner separation of concerns.

parser = HfArgumentParser((ModelArguments, DataTrainingArguments,
                           TrainingArguments))
if len(sys.argv) == 2 and sys.argv[1].endswith(".json"):
    # If we pass only one argument to the script
    # and it’s the path to a json file,
    # let’s parse it to get our arguments.
    model_args, data_args, training_args = parser.parse_json_file(
        json_file=os.path.abspath(sys.argv[1]))
else:
    model_args, data_args, training_args = parser.parse_args_into_dataclasses()

# Sending telemetry. Tracking the example usage helps us
# better allocate resources to maintain them. The
# information sent is the one passed as arguments along
# with your Python/PyTorch versions.
send_example_telemetry("run_ner", model_args, data_args)

# Setup logging
logging.basicConfig(
    format="%(asctime)s␣-␣%(levelname)s␣-␣%(name)s␣-␣%(message)s",
    datefmt="%m/%d/%Y␣%H:%M:%S",
    handlers=[logging.StreamHandler(sys.stdout)],
)

log_level = training_args.get_process_log_level()
logger.setLevel(log_level)
datasets.utils.logging.set_verbosity(log_level)
transformers.utils.logging.set_verbosity(log_level)
transformers.utils.logging.enable_default_handler()
transformers.utils.logging.enable_explicit_format()

# Log on each process the small summary:
logger.warning(
    f"Process␣rank:␣{training_args.local_rank}"
    + f"device:␣{training_args.device},␣n_gpu:␣{training_args.n_gpu}"
    + f"distributed␣training:␣{bool(training_args.local_rank␣!=␣-1)}"
    + f"16-bits␣training:␣{training_args.fp16}"
)
logger.info(f"Training/evaluation␣parameters␣{training_args}")

# Detecting last checkpoint.
last_checkpoint = None
if (os.path.isdir(training_args.output_dir) and training_args.do_train
    and not training_args.overwrite_output_dir):
    last_checkpoint = get_last_checkpoint(training_args.output_dir)
    if (last_checkpoint is None and
        len(os.listdir(training_args.output_dir)) > 0):
        raise ValueError(
            f"Output␣directory␣({training_args.output_dir})␣"
            "already␣exists␣and␣is␣not␣empty.␣"
            "Use␣--overwrite_output_dir␣to␣overcome."
        )
    elif (last_checkpoint is not None and
          training_args.resume_from_checkpoint is None):
        logger.info(
            f"Checkpoint␣detected,␣resuming␣training␣at␣{last_checkpoint}.␣"
```

```
                "To␣avoid␣this␣behavior,␣change␣"
                "the␣‘--output_dir‘␣or␣add␣‘--overwrite_output_dir‘␣"
                "to␣train␣from␣scratch."
            )

    # Set seed before initializing model.
    set_seed(training_args.seed)

    # Get the datasets: you can either provide your
    # own CSV/JSON/TXT training and evaluation files (see below)
    # or just provide the name of one of the public
    # datasets available on the hub at https://huggingface.co/datasets/
    # (the dataset will be downloaded automatically from the datasets Hub).
    #
    # For CSV/JSON files, this script will use the
    # column called ’text’ or the first column if no column called
    # ’text’ is found. You can easily tweak this behavior (see below).
    #
    # In distributed training, the load_dataset function
    # guarantee that only one local process can concurrently
    # download the dataset.
    if data_args.dataset_name is not None:
        # Downloading and loading a dataset from the hub.
        raw_datasets = load_dataset(
            data_args.dataset_name,
            data_args.dataset_config_name,
            cache_dir=model_args.cache_dir,
            use_auth_token=True if model_args.use_auth_token else None,
        )
    else:
        data_files = {}
        if data_args.train_file is not None:
            data_files["train"] = data_args.train_file
        if data_args.validation_file is not None:
            data_files["validation"] = data_args.validation_file
        if data_args.test_file is not None:
            data_files["test"] = data_args.test_file
        extension = data_args.train_file.split(".")[-1]
        raw_datasets = load_dataset(extension, data_files=data_files,
                                    cache_dir=model_args.cache_dir)
    # See more about loading any type of standard or custom dataset
    # (from files, python dict, pandas DataFrame, etc) at
    # https://huggingface.co/docs/datasets/loading_datasets.html.
    if training_args.do_train:
        column_names = raw_datasets["train"].column_names
        features = raw_datasets["train"].features
    else:
        column_names = raw_datasets["validation"].column_names
        features = raw_datasets["validation"].features

    if data_args.text_column_name is not None:
        text_column_name = data_args.text_column_name
    elif "tokens" in column_names:
        text_column_name = "tokens"
    else:
        text_column_name = column_names[0]

    if data_args.label_column_name is not None:
        label_column_name = data_args.label_column_name
    elif f"{data_args.task_name}_tags" in column_names:
        label_column_name = f"{data_args.task_name}_tags"
    else:
        label_column_name = column_names[1]

    # In the event the labels are not a ‘Sequence[ClassLabel]‘,
    # we will need to go through the dataset to get the
    # unique labels.
    def get_label_list(labels):
        unique_labels = set()
        for label in labels:
            unique_labels = unique_labels | set(label)
        label_list = list(unique_labels)
        label_list.sort()
        return label_list
    # If the labels are of type ClassLabel, they are already integers
    # and we have the map stored somewhere.
    # Otherwise, we have to get the list of labels manually.

    #labels_are_int = isinstance(features[label_column_name].feature, ClassLabel)

    #if labels_are_int:
    #    label_list = features[label_column_name].feature.names
    #    label_to_id = {i: i for i in range(len(label_list))}
    #else:
    label_list = get_label_list(raw_datasets["train"][label_column_name])
    label_to_id = {l: i for i, l in enumerate(label_list)}

    print(label_to_id)
    print(get_label_list(raw_datasets["validation"][label_column_name]))


    num_labels = len(label_list)

    # Load pretrained model and tokenizer
    #
    # Distributed training:
    # The .from_pretrained methods guarantee that only one local
    # process can concurrently
    # download model & vocab.
    config = AutoConfig.from_pretrained(
        (model_args.config_name if model_args.config_name
         else model_args.model_name_or_path),
        num_labels=num_labels,
        finetuning_task=data_args.task_name,
        cache_dir=model_args.cache_dir,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
    )

    tokenizer_name_or_path = (model_args.tokenizer_name if model_args.tokenizer_name
                              else model_args.model_name_or_path)
    if config.model_type in {"bloom", "gpt2", "roberta"}:
        tokenizer = AutoTokenizer.from_pretrained(
            tokenizer_name_or_path,
            cache_dir=model_args.cache_dir,
            use_fast=True,
            revision=model_args.model_revision,
            use_auth_token=True if model_args.use_auth_token else None,
            add_prefix_space=True,
        )
    else:
        tokenizer = AutoTokenizer.from_pretrained(
            tokenizer_name_or_path,
            cache_dir=model_args.cache_dir,
            use_fast=True,
            revision=model_args.model_revision,
            use_auth_token=True if model_args.use_auth_token else None,
        )

    model = AutoModelForTokenClassification.from_pretrained(
        model_args.model_name_or_path,
        from_tf=bool(".ckpt" in model_args.model_name_or_path),
        config=config,
        cache_dir=model_args.cache_dir,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
        ignore_mismatched_sizes=model_args.ignore_mismatched_sizes,
    )

    # Tokenizer check: this script requires a fast tokenizer.
    if not isinstance(tokenizer, PreTrainedTokenizerFast):
        raise ValueError(
            "This␣example␣script␣only␣works␣for␣models␣that␣have␣a␣"
            "fast␣tokenizer.␣Checkout␣the␣big␣table␣of␣models␣at"
            "␣https://huggingface.co/transformers/index.html#supported-frameworks␣"
            "to␣find␣the␣model␣types␣that␣meet"
            "␣this␣requirement"
        )

    # Model has labels -> use them.
    if model.config.label2id != PretrainedConfig(num_labels=num_labels).label2id:
        if list(sorted(model.config.label2id.keys())) == list(sorted(label_list)):
            # Reorganize ‘label_list‘ to match the ordering of the model.
            label_list = [model.config.id2label[i] for i in range(num_labels)]
            label_to_id = {l: i for i, l in enumerate(label_list)}
        else:
            logger.warning(
                "Your␣model␣seems␣to␣have␣been␣trained␣with␣labels,␣but␣they␣"
                "don’t␣match␣the␣dataset:␣",
                f"model␣labels:␣{list(sorted(model.config.label2id.keys()))},␣"
                "dataset␣labels:"
                f"␣{list(sorted(label_list))}.\nIgnoring␣"
                "the␣model␣labels␣as␣a␣result.",
            )

    # Set the correspondences label/ID inside the model config
    model.config.label2id = {l: i for i, l in enumerate(label_list)}
    model.config.id2label = {i: l for i, l in enumerate(label_list)}

    # Map that sends B-Xxx label to its I-Xxx counterpart
    b_to_i_label = []
    for idx, label in enumerate(label_list):
        if label.startswith("B-") and label.replace("B-", "I-") in label_list:
            b_to_i_label.append(label_list.index(label.replace("B-", "I-")))
        else:
            b_to_i_label.append(idx)

    # Preprocessing the dataset
    # Padding strategy
    padding = "max_length" if data_args.pad_to_max_length else False
```

```
# Tokenize all texts and align the labels with them.
def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(
        examples[text_column_name],
        padding=padding,
        truncation=True,
        max_length=data_args.max_seq_length,
        # We use this argument because the texts in our dataset
        # are lists of words (with a label for each word).
        is_split_into_words=True,
    )

    labels = []
    for i, label in enumerate(examples[label_column_name]):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        previous_word_idx = None
        label_ids = []
        for word_idx in word_ids:
            # Special tokens have a word id that is None. We set
            # the label to -100 so they are automatically
            # ignored in the loss function.
            if word_idx is None:
                label_ids.append(-100)
            # We set the label for the first token of each word.
            elif word_idx != previous_word_idx:
                if not label[word_idx] in label_to_id:
                    print("Label␣" + label[word_idx]
                          + "␣is␣not␣in␣training␣data")
                    label_ids.append(label_to_id['O'])
                else:
                    label_ids.append(label_to_id[label[word_idx]])
            # For the other tokens in a word, we set the label to
            # either the current label or -100, depending on
            # the label_all_tokens flag.
            else:
                if data_args.label_all_tokens:
                    label_ids.append(b_to_i_label[label_to_id[label[word_idx]]])
                else:
                    label_ids.append(-100)
            previous_word_idx = word_idx


        labels.append(label_ids)
    tokenized_inputs["labels"] = labels
    print(tokenized_inputs[0])
    return tokenized_inputs

if training_args.do_train:
    if "train" not in raw_datasets:
        raise ValueError("--do_train␣requires␣a␣train␣dataset")
    train_dataset = raw_datasets["train"]
    if data_args.max_train_samples is not None:
        max_train_samples = min(len(train_dataset), data_args.max_train_samples)
        train_dataset = train_dataset.select(range(max_train_samples))

    with training_args.main_process_first(
        desc="train␣dataset␣map␣pre-processing"):
        train_dataset = train_dataset.map(
            tokenize_and_align_labels,
            batched=True,
            num_proc=data_args.preprocessing_num_workers,
            load_from_cache_file=not data_args.overwrite_cache,
            desc="Running␣tokenizer␣on␣train␣dataset",
        )

if training_args.do_eval:
    if "validation" not in raw_datasets:
        raise ValueError("--do_eval␣requires␣a␣validation␣dataset")
    eval_dataset = raw_datasets["validation"]
    if data_args.max_eval_samples is not None:
        max_eval_samples = min(len(eval_dataset), data_args.max_eval_samples)
        eval_dataset = eval_dataset.select(range(max_eval_samples))
    with training_args.main_process_first(
        desc="validation␣dataset␣map␣pre-processing"):
        eval_dataset = eval_dataset.map(
            tokenize_and_align_labels,
            batched=True,
            num_proc=data_args.preprocessing_num_workers,
            load_from_cache_file=not data_args.overwrite_cache,
            desc="Running␣tokenizer␣on␣validation␣dataset",
        )

if training_args.do_predict:
    if "test" not in raw_datasets:
        raise ValueError("--do_predict␣requires␣a␣test␣dataset")
    predict_dataset = raw_datasets["test"]
    if data_args.max_predict_samples is not None:
        max_predict_samples = min(len(predict_dataset),
                                  data_args.max_predict_samples)
        predict_dataset = predict_dataset.select(range(max_predict_samples))
    with training_args.main_process_first(
        desc="prediction␣dataset␣map␣pre-processing"):
        predict_dataset = predict_dataset.map(
            tokenize_and_align_labels,
            batched=True,
            num_proc=data_args.preprocessing_num_workers,
            load_from_cache_file=not data_args.overwrite_cache,
            desc="Running␣tokenizer␣on␣prediction␣dataset",
        )

# Data collator
data_collator = DataCollatorForTokenClassification(
    tokenizer,
    pad_to_multiple_of=8 if training_args.fp16 else None)

# Metrics
metric = evaluate.load("seqeval")

def compute_metrics(p):
    predictions, labels = p
    predictions = np.argmax(predictions, axis=2)

    # Remove ignored index (special tokens)
    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    results = metric.compute(
        predictions=true_predictions,
        references=true_labels)
    if data_args.return_entity_level_metrics:
        # Unpack nested dictionaries
        final_results = {}
        for key, value in results.items():
            if isinstance(value, dict):
                for n, v in value.items():
                    final_results[f"{key}_{n}"] = v
            else:
                final_results[key] = value
        return final_results
    else:
        return {
            "precision": results["overall_precision"],
            "recall": results["overall_recall"],
            "f1": results["overall_f1"],
            "accuracy": results["overall_accuracy"],
        }

# Initialize our Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset if training_args.do_train else None,
    eval_dataset=eval_dataset if training_args.do_eval else None,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

# Training
if training_args.do_train:
    checkpoint = None
    if training_args.resume_from_checkpoint is not None:
        checkpoint = training_args.resume_from_checkpoint
    elif last_checkpoint is not None:
        checkpoint = last_checkpoint
    train_result = trainer.train(resume_from_checkpoint=checkpoint)
    metrics = train_result.metrics
    trainer.save_model()  # Saves the tokenizer too for easy upload

    max_train_samples = (
        data_args.max_train_samples if data_args.max_train_samples is not None
        else len(train_dataset)
    )
    metrics["train_samples"] = min(max_train_samples, len(train_dataset))

    trainer.log_metrics("train", metrics)
    trainer.save_metrics("train", metrics)
    trainer.save_state()

# Evaluation
if training_args.do_eval:
    logger.info("***␣Evaluate␣***")
    metrics = trainer.evaluate()

    max_eval_samples = (
        data_args.max_eval_samples if data_args.max_eval_samples is not None
        else len(eval_dataset)
    )
    metrics["eval_samples"] = min(max_eval_samples, len(eval_dataset))

    trainer.log_metrics("eval", metrics)
    trainer.save_metrics("eval", metrics)

# Predict
if training_args.do_predict:
    logger.info("***␣Predict␣***")

    predictions, labels, metrics = trainer.predict(
        predict_dataset,
        metric_key_prefix="predict")
    predictions = np.argmax(predictions, axis=2)


    # Remove ignored index (special tokens)
    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    trainer.log_metrics("predict", metrics)
    trainer.save_metrics("predict", metrics)

    # Save predictions
    output_predictions_file = os.path.join(
        training_args.output_dir, "predictions.txt"
        )
    if trainer.is_world_process_zero():
        with open(output_predictions_file, "w") as writer:
            for prediction in true_predictions:
                writer.write("␣".join(prediction) + "\n")

kwargs = {"finetuned_from": model_args.model_name_or_path,
          "tasks": "token-classification"}
if data_args.dataset_name is not None:
    kwargs["dataset_tags"] = data_args.dataset_name
    if data_args.dataset_config_name is not None:
        kwargs["dataset_args"] = data_args.dataset_config_name
        kwargs["dataset"] = (f"{data_args.dataset_name}␣"
        + f"{data_args.dataset_config_name}")
    else:
        kwargs["dataset"] = data_args.dataset_name

if training_args.push_to_hub:
    trainer.push_to_hub(**kwargs)
else:
    trainer.create_model_card(**kwargs)


def _mp_fn(index):
# For xla_spawn (TPUs)
main()
```

```


if __name__ == "__main__":
    main()
```

Appendix F

Performance Script

Code listing F.1: Script that computes the performance of the individual entity tytpes

```
import itertools
import evaluate
import json
from sklearn import metrics

prediction_file = open('result-model_5_20/predictions.txt', 'r')

predictions = prediction_file.read()

predictions = predictions.split('\n')

predictions = list(map(lambda p: p.split('␣'), predictions))


true_file = open('data/test_5.json')

true_data = json.load(true_file)

true = []

for sentence in true_data:
   true.append(sentence['ner_tags'])

predictions = list(filter(lambda p: p != [''], predictions))
true = list(filter(lambda t: t != [], true))

for i in range(len(true)):
   if not len(true[i]) == len(predictions[i]):
      print(true[i])


seqeval = evaluate.load('seqeval')
results = seqeval.compute(predictions=predictions, references=true)

print(results)
```