University of Zurich^UZH

Thesis February 1, 2022

Style Transfer Algorithm for Online News

Lukas Yu
of University of Zurich, Switzerland

Student-ID: 14-720-866
lukas.yu@uzh.ch

Advisor: **Lucien Heitz**
Co-advisor: **Annette Rios**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Universität Zürich
http://www.ifi.uzh.ch/ddis

Acknowledgements

First and foremost, I would like to thank my advisor Lucien Heitz, who continuously assisted me throughout this thesis. This thesis would not have been possible without his invaluable guidance and support.

I must also thank my co-advisor Annette Rios, who graciously offered her expert knowledge in computational linguistics and provided a code base to start out with.

Finally, I would like to express my gratitude to my girlfriend, Alice Truong, for proof-reading and my family for supporting me in every way they could.

Zusammenfassung

In wissenschaftliche Experimente ist es äusserst wichtig, alle relevante Daten gründlich zu anonymisieren. Für Experimente mit Newsartikeln ist es besonders schwierig, sie effektiv von ihren Quellen zu entkoppeln. Versuchspersonen könnten anhand des Schreibstils erkennen, von wo sie stammen. Diese Arbeit befasst sich damit, mit modernen neurale Netzen Texte in einem einheitlichen Stil zu transformieren. Die neue Methode benötigt keine parallele Datensätze, was üblicherweise die grösste Einschränkung für neurale Netze ist. Ein pseudo-paralleles Korpus wird mit einfachen Artikeln mithilfe von Masked-language Modelling erstellt. Zusätzlich wird ein neues Scrapersystem entworfen und implementiert, um Artikel von Webseiten von Zeitungen zu extrahieren und sie in einem homogenen Format abzuspeichern.

Abstract

In an experimental setting, data anonymization is vital to get valid results. For studies dealing with news articles, white-labelling their source is a non-trivial task, since news outlets might possess traceable writing styles. In this thesis, modern neural network architectures for natural language processing are utilized to transfer texts to a uniform style. The method does not rely on parallel corpora, which is usually the bottleneck for many systems. Instead, a pseudo-parallel corpus is created using monolingual data and masked-language modeling. Additionally, a new scraper architecture is designed and implemented to easily obtain article from news websites and store them in a homogeneous format.

Contents

1 Introduction

Writing style is an important aspect of conveying information from the writer to the reader, regardless of the actual content. Different characteristics, such as simplicity, politeness, persuasiveness, framing and many others have a significant impact on how the reader will perceive the narrative. The writing style of a text will primarily depend on the intent of the writer for the particular purpose it is supposed to serve, but it is also influenced subconsciously by the context of the situation.

This is particularly apparent in news articles. Multiple news outlets will report the same event, but will rarely use the same writing style. Moreover, articles from the same news outlet will show similar writing styles. News outlets typically have a writing style across their publications to cater towards a particular audience, who in turn have expectations on how the articles are presented. A reader who is well accustomed to read news articles from certain news outlets may even guess from which news outlet an article originated solely based on how the article is written. This is especially problematic in scientific experiments where researchers might not want to disclose the source of news articles.

Informfully is an application built by the DDIS group of University of Zurich. The *Informfully* system is a research platform designed for conducting experiments investigating the behavior of participants reading news articles on mobile devices. It is purposefully designed to be as customizable as possible to enable a wide variety of studies. The system has four main components: the admin website, the recommender system, the mobile app and the scraper pipeline.

Researchers can set up an experiment on the admin website and create the desired amount of user accounts and surveys. The credentials of the user accounts can then be distributed to the participants, who are asked to download the *Informfully* app available for Android and iOS devices from the Google Play Store or Apple App Store respectively. Upon logging into the app for the first time, an initial survey is presented to the user. An individual profile is created for each user based on the results of that survey, which serves as input for the recommender system. The recommender system generates a customized list of articles for every user from the database of articles obtained from the scrapers. This list is then shown to the user as a news feed on their home page in the app. Its algorithm can also be altered by the researchers to suit their studies.

Participants are expected to open the app at least once per day and read any articles of interest to them. After reading an article, a short custom survey is shown to the participants to assess their opinion on that specific article. This additional information is also taken into consideration by the recommender system for future recommendations. Along with the surveys, other data on interactions with the app are also logged such as opening and closing an article, how long an article was open, how long the app itself was open etc.

To ensure the integrity of the results stemming from experiments, the source of the articles have to be white-labelled, i.e., the information from which news outlet a news article comes from will not be disclosed to the participant. In the current version of the system, the name of the news outlet is hidden from the user. In addition to that, any occurrence names of news outlets inside the article text will be replaced with an anonymous token.

However, during a recent pilot experiment, concerns were raised that participants could guess the source of an article based on the formatting or the writing style. To get around this issue is anything but a trivial problem.

The idea of text style transfer is to rewrite a text to another style, while preserving the original content. Usually, one wants to use text style transfer to precisely control style attributes, for example politeness. In this case however, the goal is not to manually set individual attributes, but to rewrite articles to a specific writing style of a news outlet in order to generate genuinely white-labelled articles. The goal of this thesis is to create a sequence-to-sequence model to transfer writing styles of news articles.

1.1 Thesis Structure

This work is split into two separate parts: 1.) text style transfer and 2.) new article scraper. The main focus of this paper is on the text style transfer, while the scraper can be treated as a side project. For this reason, the main content covers the work related to text style transfer, while the documentation on the scraper work is moved to appendix A.

The text style transfer part is further split into four parts. The first is a background chapter, where the necessary background knowledge is provided to the reader to fully understand the remainder of the thesis. It briefly explains the development of the Transformer model and its derivatives during the last few years. It is expected that the reader possesses some level of expertise regarding neural networks.

The second chapter covers the methodology used to create the text style transfer model for online news. Every step involved in the process, as well as the thought processes behind them will be explained in detail.

In the third chapter, the setup of the user study to investigate the extent a reader is able to recognize writing style is explained first. The results are discussed directly afterwards.

For the last chapter, there will be a summary of the thesis and some words to conclude the main work.

2

Background

Deep learning has long been the best performing method in the general field of Natural Language Processing (NLP) compared to traditional non-neural methods. Throughout this chapter, a focus will be set on text sequence-to-sequence tasks such as translation, response generation or the most relevant for now: Text Style Transfer (TST). In recent years, research on TST have been active with a considerably high number of different approaches for a wide variety of applications [Jin et al., 2020]. In the upcoming sections, background knowledge of a few selected technologies will be provided to give context about the current scientific landscape of TST. It will also aid the reader to better understand the remaining parts of this thesis.

2.1 Transformer Architecture

The introduction of the transformer architecture [Vaswani et al., 2017] is likely one of the larger contributor of the latest success. Compared to architectures following the designs of Recurring Neural Networks (RNN) such as Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] or Gated Recurrent Unit (GRU) [Cho et al., 2014], which were previously the most popular choices for sequence processing, transformers can be trained faster and simultaneously yield better results.

Sequence-to-sequence models usually work with an encoder/decoder architecture. The encoder takes the input text and converts its meaning into a vector representation. The decoder then takes the vector and generates the desired output text depending on the task.

With RNNs, each word in a sentence is fed to the encoder piece by piece. To process the next word in an input sequence, the output of the model from the previous step is required. Decoding works similarly, one word at a time and requiring the previously generated word to generate the next word. Both encoding and decoding with this approach cannot be parallelized, meaning that handling large amounts of data will be very inefficient. Another limitation with RNNs is the context vector, the last output vector the encoder creates after the final word in a sequence has been processed. Ideally, the input sentence would be perfectly summarized inside the context vector for the decoder. In practice, however, there is only a finite amount of information that a fixed size vector can describe. Especially with longer sentences, the model struggles to preserve all the information. The *attention* mechanism [Bahdanau et al., 2015] attempts to solve this issue by not only considering the context vector at the end, but also the intermediate output vectors after each word. While decoding, at each step, the attention mechanism will calculate an attention score for each context vector based on how relevant it is to the current step, which then is used to create a weighted average context vector. As a result, more relevant states will have a greater influence on the output word. In other words, the decoder is able to *attend* to a list of context vectors to pick out the more suited states for the current position.

The transformer architecture adopts the idea of attention, but implements it in a different way. Using *self-attention*, the transformer detects how relevant one word is in a sentence in regard to the other words of the same sentence. Because transformers do not have any recurrent components, it does not exhibit any limitations of the context vector and its operations can be completely parallelized, significantly reducing the time for training and inference with state-of-the-art performance.

2.2 BERT and Variants

Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2019] is a language representation model. Its goal is to read and understand a text and create embeddings representing the meaning of texts. The architecture is very similar to the encoder of the original transformer. However, BERT is not just an architecture of a model, but rather a pre-training technique. A pre-trained BERT model will create embeddings that can be used for further a variety of downstream tasks such as translation, question answering, sentiment analysis etc. Although it is not a sequence-to-sequence model by itself, its powerful encoding capabilities can be used in conjunction with a separate decoder to generate text. Fully training a BERT model has two steps: pre-training and fine-tuning.

Pre-training a BERT model works by having two training objectives: mask filling and next sentence prediction. Both objectives are learned using only non-parallel data, that is simple text without source and target alignment. Mask filling works by replacing some words in the input sentence with a mask token. The model then tries to predict the masked tokens. With next sentence prediction, the model is given two sentences and decides if the second comes after the first to learn relationships between sentences. A large selection of pre-trained BERT models in English are available on the Github page of the paper[1], therefore it is usually recommended to start from a pre-trained model, as pre-training a decent model typically requires large amount of resources and time. Not relying on true parallel data to perform pre-training is a substantial advantage, as monolingual data in large quantities is more easily obtainable.

Transforming a pre-trained BERT model to perform a specific task will require fine-tuning. The pre-trained model will already have a good understanding of language, so what fine-tuning accomplishes is first to replace the last layer to match the desired

[1]Code repository of BERT: *https://github.com/google-research/bert*

output format, then using the task specific parallel data to just train the last layer. Fine-tuning is significantly less resource intensive compared to pre-training.

At the time of introduction of BERT, it obtained state-of-the-art performance for numerous NLP tasks. Its success leads to a host of variants and extensions of this model to accommodate other needs and tasks. A small selection of examples includes RoBERTA [Liu et al., 2019], ALBERT [Lan et al., 2019], ELECTRA [Clark et al., 2020], DistilBERT [Sanh et al., 2019] and BERTSUM [Liu, 2019].

2.2.1 BART

Unlike BERT, BART [Lewis et al., 2019] is a fully sequence-to-sequence architecture with both encoder and decoder. It can be considered a generalization of the BERT encoder and Generative Pre-trained Transformer (GPT) [Radford and Narasimhan, 2018] as encoder. Its architecture is comparable with the original transformer again. However, the innovation comes from the new noising techniques introduced for pre-training. Instead of only token masking and next sentence prediction, BART includes token masking together with other noising schemes like token deletion, text infilling, sentence permutation and document rotation.

The BART paper [Lewis et al., 2019] presents evaluations on common benchmark tasks and its proposed model shows similar or better results as other leading models.

2.2.2 mBART

mBART [Liu et al., 2020] is a multilingual sequence-to-sequence denoising autoencoder specialized for language translation. Just like BART, its architecture resembles the original transformer. It also utilizes the same pre-training objectives as BART. However, while BERT or BART mainly focused on using the models for one language, usually English, mBART uses multiple languages for pre-training instead. Using a large-scale multilingual corpus consisting of 25 languages, the model is pre-trained once for every language. The model can then be fine-tuned for a specific language pair (e.g., Spanish $\rightarrow$ English) with parallel data to get a translation model. This approach already shows improvements for language pairs where parallel data is scarce, only losing in comparison to other state-of-the-art methods when both given a large amount of parallel data. In addition to that, mBART is able to improve performance on languages that were not present during pre-training, which indicates that the model learns some aspects that are language universal.

A model pre-trained on 25 languages is available on this repository.[2]

2.2.3 Author Style Rewriting

StyleLM [Syed et al., 2019] is an approach to implement author-specific style rewriting with an architecture similar to mBART. It is first pre-trained on a large English corpus

[2]Code repository of mBART: *https://github.com/pytorch/fairseq/blob/main/examples/mbart/README.md*

and then fine-tuned for each author individually. The resulting models generate rewritten texts from any input and tries to emulate the writing style of that author.

2.3 Byte Pair Encoding

Byte pair encoding (BPE) [Gage, 1994] was initially introduced as a data compression algorithm. It works by finding the most common byte pair and replacing them with a single byte that does not occur in the initial data, while recording every step. The newly added bytes can also be replaced in pairs recursively. Reversing the process will decompress the data to its original state.

A limitation of text generation neural networks is the fixed vocabulary size. A predetermined dictionary of words must be defined prior to model training. This poses a problem, since many languages are open ended, meaning a theoretical unlimited number of words are possible due to word formation processes such as compounds. In context of NLP, BPE is used to create a sub word vocabulary [Sennrich et al., 2016] for word tokenizing. By adapting the algorithm to keep the byte pair intact instead of replacing them, a vocabulary of sub words can be constructed. The algorithm is applied recursively until a specified amount of sub words is reached.

Using sub words to tokenize the input text, rarer words that would otherwise not be included in a traditional vocabulary can now be deconstructed into multiple sub words. With the same vocabulary size, the tokenizer is now able to cover a larger number of words. To showcase an example, the German word *Feuerwehrzufahrt* is split up into *Feuerwehr*, *zu* and *fahrt*. It should be noted that this is an unsupervised algorithm, therefore only learning from raw text data and not by any logic based on language grammar. As a result, compound words may not always correctly split into its lexemes.

3

Methodology

In this chapter, the exact procedures of every step from data selection up to a finished style transfer model will be explained to be as reproducible as possible. First, a general explanation for the approach, text style transfer, is given. After that, the next sections go into more detail about the process of building the model. Starting with the collection and pre-processing of raw data, followed by the architecture choice, the creation of a pseudo-parallel corpus and the training.

Initially, a simplification model was considered. The core idea was to bring the complexity of every article down to a uniform level, based on the assumption that most differences in writing styles between outlets only stem from a difference in complexity. However, there are some issues with this method. Writing styles are presumably too intricate to be described with a simple and one-dimensional complexity attribute. Two texts with equal complexity can still be discerned regarding style by other characteristics such as word choice, sentence structure as well as more abstract ones like formality. Another problem is the availability of suitable parallel data. Simplification datasets in German are usually sourced from government sites, where simplified versions of the text are required by law. Not only are such datasets sparse, but the type of text is also profoundly different compared to news articles. Using text data from another domain could negatively impact the results.

In the end, a decision was made to pursue another concept, text style transfer. The goal of this approach is to white-label a group of article texts by rewriting each text while transferring them into a uniform style. Since there are no parallel data available, masked language modeling is used to create a noisy version of the text, which is then fed to the model, who tries to restore the original text as training.

3.1 Data Source and Pre-Processing

The training of neural networks typically requires large amounts of data to be successful. In this case, the data consists of text from news articles written in different writing styles to learn from.

The corpus consists of approximately 1.2 million articles from a variety of Swiss news outlets written in German. An overview of the distribution of articles can be seen in table 3.1

News Outlet	**Article Count**
Neue Zürcher Zeitung	313,295
Schweizer Radio und Fernsehen	261,253
Tages-Anzeiger	251,399
Blick	154,433
NZZ am Sonntag	62,148
Tages-Anzeiger SonntagsZeitung	49,927
SonntagsBlick	47,590

Table 3.1: Number of articles for each news outlet

The publication dates of these articles range from January 2012 up to February 2021, ordered decreasingly by article count.

A crucial step before starting the training is the pre-processing of data. If the raw data were directly fed to the learning model, the results would be suboptimal. The initial data must be processed to elevate its data quality to a usable state.

When looking at the archive of news articles, it quickly becomes apparent that not a lot of effort has gone into ensuring a high level of data quality. Since it is a dump of all content from the databases of the news outlets, a lot of the texts have formatting errors. Unwanted anomalies, such as missing spaces, random newline characters, twitter and image links, image captions and source information are common throughout the archive. In addition to that, due to the number of different source news outlets, which all have their own formatting, one pre-processing method that might work on articles from one outlet might not be perfectly applicable to another. Thus, special attention must be given to each of the outlets to provide the most optimal foundation for further processing.

The data were given in the format of CSV files, one file for each news outlet. To enable to possibility to query the articles in a more complex manner, the CSV entries are imported into a MongoDB database, which was already set up from working on the Informfully infrastructure. All the fields are preserved, with an addition of an *outlet* field to denote the source outlet.

The remaining pre-process is handled by a Jupyter Notebook[1] script. The goal in the end is to create a list of labeled sentences that is ultimately split into three target files: *train.tgt*, *val.tgt* and *test.tgt*. The purpose of the training data is to be used to fit the model, whereas the validation data is used during training to frequently test its performance on it. Based on the test results on the validation data, the training hyper-parameters will be adjusted. After training has been completed, the model will be tested against the test data to give an unbiased assessment of its performance.

The process to create the three files starts from the articles stored in the database. The first step is to select a subset of articles. If the entire dataset were to be used, training could not be finished in a reasonable amount of time. An assumption was made that the writing style may change, especially over a longer time span. The most recent writing

[1]Jupyter website: *https://jupyter.org/*

style is likely to be the desired style the model should be able to replicate. Therefore, the articles are limited to the latest 1,000 for each news outlet.

In the next step, the article text, which are large, continuous strings of text is split into separate sentences. The NLTK[2] sentence tokenizer for German is utilized for this task. Using a more sophisticated method of breaking up sentences instead of splitting the text by a period or other sentence ending punctuation is necessary to get consistent results due to words such as name initials (J.S.), titles (Dr.) or abbreviations of geographical places (St. Gallen). At the end of this step, a list of all sentences is ready for further processing.

By taking a glance at the sentences, it becomes clear that a lot of the sentences are not proper sentences, but rather text from formatting, data tables, web links and other non-continuous text. To filter these lines out of the data, a length comparison is done. If the line is shorter than 20 characters or longer than 100 characters, it will be excluded. This simple solution is quite effective, as it removes almost every unwanted lines. Any newline character (\n) is also removed, as it is not necessary for a sentence-based corpus.

The lines are now split into a train, a development, and a test set with a respective percentage of 80%, 10% and 10%. The lines all follow the format of

```
<outlet_tag> This is an example of a line.
```

The first token, the <outlet_tag> is a tag denoting the outlet which published the article which included that line.

3.2 Non-Parallel Corpora and Masked Language Modeling

Supervised machine learning works by having a source and target pair for each sample as data. The source being the input for the model and the target being the result the model should ideally produce given the input. By adjusting the model to minimize the difference between the target and the actual output, the model slowly learns to improve its performance. A collection of text pair samples is called a parallel corpus.

Applying this method to the current task of style transfer, samples of two sentences describing the same content, but in different writing styles are required. However, such data is rarely easily available in sufficient quantities. While there are certainly articles about the same topic from multiple outlets, they are seldom useful. In most cases, the articles are directly taken from new agencies with minimal adjustments. As a result of that, the articles are virtually identical.

Manually producing pairs of sentences would require someone to be well versed in the language, be able to write in different styles and have an unreasonable amount of time to translate. In short, it is safe to assume that this method of producing a parallel corpus is infeasible.

To overcome this problem, masked-language modeling, similar to the method used in BERT [Devlin et al., 2019], is used for training. With this type of training, a monolingual corpus is already sufficient. The existing data is used as target data, whereby the

[2]NLTK website: *https://www.nltk.org/*

source data is created from that target data by introducing noise into the sentence. The model then attempts to recreate the original sentence in order to learn with the usual deep learning procedure.

The noise added to the sentences takes form of randomly switching out tokens with *mask* tokens. The model will learn to *fill the gaps* with the right words that are appropriate for the desired writing style.

In practice, an input sentence text is first tokenized into token ids using *sentencepiece*[3] to apply BPE. In the next step, each input ID has a 15% chance to be replaced with the id of the mask token <MASK>. This partially masked list of token ids is then fed into the model as the source sentence, with the full and unmasked sentence as the target.

The implementation of the model makes heavy use of the *Huggingface Transformers*[4], a Python library to simplify the application of machine learning models. Together with its *model hub*,[5] which offers pre-trained models directly accessible via Python, it makes a powerful tool to use and train models in a short time span.

An mBART model (see chapter 2) is trained to learn and apply TST. The pre-trained model *facebook/mbart-large-cc25* is downloaded from the Huggingface model hub. This mBART model is pre-trained on 25 languages, including German, making it the excellent choice for this use case. Before the model is fine-tuned, the vocabulary size is decreased to 25,000 sub word tokens, since only German text is processed. A list of the most frequent tokens is generated by applying the existing tokenizer from the pre-trained model and applying them to the article texts. The 25,000 tokens that occurred the most in the data are kept, while the others are trimmed out. A smaller vocabulary size decreases the complexity of the model and leads to faster training as well as inference.

In the end, the model has to accept a label conveying which style the text should be transferred to. The mBART model was used for initally used multi-language processing, but in this case the target language, i.e., German, stays fixed during the entire process. However, the multi-language mechanism will be *misused* to also denote writing styles. For each news outlet, a new language tag is added by copying the same embedding as the German tag *de_DE*, enabling training towards a specific outlet style. An additional tag *de_NOISE* with German initialization is added for training purposes in order to preserve the normal German tag in its original state that will be used for testing. Then, with fine-tuning, the model should learn the defining characteristics of each style. The other language tags will not be used for the remainder of this work.

3.3 Setup, Training and Application

Training was performed using following hyperparameters: maximum input and output length of 512, batch size of 8, gradient accumulation of 4, attention dropout of 10%, standard dropout of 30%, label smoothing with 0.2, learning rate of 3^{-5}, validation loss

[3]Code repository of sentencepiece: *https://github.com/google/sentencepiece*
[4]Huggingface Transformers website: *https://huggingface.co/docs/transformers/index*
[5]Huggingface Model Hub: *https://huggingface.co/models*

as early stopping metric, patience of 5, learning rate reduction patience of 8 and learning reduction factor of 0.5.

The first attempt to train a model was on a local computer with a NVIDIA GeForce GTX 1070, but memory ran out with 8 GB of VRAM, resulting in an error and consequently stopping the training process. Shortly after requesting an account for a shared server from the university, training could be started without an instant error. The server is equipped with an Intel Xeon E5-2650 v4 CPU, 128 GB of RAM and seven NVIDIA GeForce GTX TITAN X with 12 GB of VRAM each.

The code to perform the modifications on the pre-trained model as well as for the training resides on a private Gitlab repository hosted on servers by the University of Zurich. The majority of the code base is forked from a different repository[6] owned by the co-supervisor of this thesis. Most notably from the *readvisor* branch written by a colleague. The only substantial extension is a separate script to add language tags to the model and the data pre-processing scripts.

The original purpose of the Longformer repository was to implement the Longformer [Beltagy et al., 2020] windowed attention into an mBART [Liu et al., 2020] model for simplification tasks. Within this thesis, the longformer functionality was not utilized at all, only the mBART component. Compared to using another code base, using longmbart had the advantage of having a co-supervisor who is familiar with the code and was able to help with technical problems.

3.4 Further Improvement Attempts

At the beginning of the project, a relatively simple approach was implemented to quickly obtain first results. It was important that a working prototype capable of training and applying text style transfer was ready, eliminating any technical problems in the process before even more complex approaches were considered. Getting high quality results was only a secondary goal at first. The text sections preceding this part explained the basic methodology of the text style transfer mode. In this section, additional methods are described that were implemented to raise the performance of the model.

3.4.1 Whole Word Masking

Subword tokenization with BPE [Sennrich et al., 2016] breaks up input words into smaller parts to decrease the necessary vocabulary size. However, this method of tokenizing creates an issue with the process of token masking. Subword tokens are individually masked randomly with a pre-determined chance. A consequence of both methods being applied simultaneously is the possibility of a whole word only being masked partially. This happens when a word is first divided into multiple subwords and then being only masked partially. A word such as *verstanden* may be split into *ver*, *stand* and *en*. If only the last subword is masked, the model would easily guess the missing piece by only looking at

[6]Code repository of longmbart: *https://github.com/a-rios/longmbart*

the other parts. In this situation, the model would barely learn from the context of the entire sentence. A better solution is to mask all subwords that belong to the same word at once. This improvement was first implemented in an update to the BERT source code[7].

3.4.2 Deletion

Another improvement approach that was implemented was to delete tokens outright, rather than just replace them with a mask token. The purpose of this method is to further add noise to the text. Just like token masking, this step is accomplished during the data pre-processing phase. According to the BART paper [Lewis et al., 2019], deletion outperforms even token masking for generation tasks.

3.4.3 Multi-Sentence Based Translation

The initial model was set up to process single sentences. However, some relevant information could reside between sentences that could help improve the style transfer. To take more sentences into context, training data now consists of three sentences per sample.

3.4.4 Additional Data

The results with the model, even with the additional improvement attempts were not satisfactory. A last attempt to get better results was to increase training data to 100,000 articles in total.

3.5 Results

First results before the additional improvement attempts were not great. The output of any transferred text was almost identical the same as the input. Only minor differences could be found that did not contribute to a change of style.

Despite numerous attempts to improve the performance of the model, the results unfortunately did not improve. An evaluation to assess the model was omitted for this reason.

[7]Code repository of BERT: *https://github.com/google-research/bert*

4

User Study

The thesis was started based on the notion that readers of news articles could guess the source based on the writing style, compromising results of scientific experiments where participants are not supposed to know the news outlet. This concern was raised from media experts and seemed plausible. This eventually led to the idea of normalizing the writing style of articles for white-labelling.

However, the extent of what an average reader is able to notice has not been explored yet. A user study is set up with the goal to evaluate the ability of its participants to distinguish writing styles.

It may even be a possibility that news outlets used in the Informfully app do not possess an inherent writing style that could be identified, or at least not a significant one an average reader or a model could recognize. In that case, articles used within an experiment with Informfully do not need special processing to obscure their origin. The current approach of simply removing obvious references revealing the source would already be sufficient. And in the case such differences do in fact exist, the user experiment can help to identify aspects of articles contribute the most to recognizability. This data can then be taken into consideration for future improvements.

4.1 Experiment Setup

Qualtrics[1] was used to create the surveys, distribute them to the respondents as well as to gather the results of the study with 11 participants.

In the survey, participants are shown pairs of shortened versions of articles from either the same or from different news outlets. After reading both texts, they are asked whether they believe the sources of the two articles differ based on writing style. Participants are then presented with a single choice question with three answer possibilities:

- Ähnlich (aus der gleichen Zeitung)
- Unterschiedlich (aus verschiedene Zeitungen)
- Kann ich nicht beurteilen

[1]Qualtrics website: *https://www.qualtrics.com/*

	SRF	**BLICK**	**TA**	**NZZ**
SRF	2	2	2	2
BLICK		2	2	2
TA			2	2
NZZ				2

Table 4.1: Table showing the amount of article pairs for each news outlet pairing with: Schweizer Radio und Fernsehen (SRF), Blick (BLICK), Tages-Anzeiger (TA) and Neue Zürcher Zeitung (NZZ).

In English, the three answers corresponds to:

- Similar writing style (same news outlet)
- Different writing style (different news outlet)
- Unable to answer

To formally define the hypothesis, consider the null hypothesis H_0 as "*a participant is not able to distinguish news outlets by the writing style.*" In that case, the chance of a participant correctly answering the question is 50% ($H_0 : P(correct) = 0.5$). The alternative hypothesis H_a is the inverse of the null hypothesis: "*a participant is able to distinguish them,*" i.e., a chance other than 50% ($H_a : P(correct) \neq 0.5$).

The articles are chosen from the same archive that was used in the text style transfer training in chapter 3. Articles from the new scrapers were also considered, but since the new scrapers were implemented during this thesis, all the scraped articles were relatively new. To avoid participants recognizing any articles and therefore invalidating the results, only articles that were at least a year old were chosen from the archive.

Out of the seven outlets, three are excluded for being Sunday editions of news outlets[2]. To ensure a more consistent articles content-wise, only articles from the category *Economy* are considered. A total of 40 articles are selected, 10 articles from each news outlet. Then, with those 40 articles, 20 article pairs are formed. For every outlet combination, including the same outlet twice, two article pairs are formed. With this setup, the two articles from 8 pairs are from the same outlet, and 12 pairs from different ones. The order of questions, as well as the two articles of each question are randomized.

The distribution of article pairs is summarized in table 4.1. Not every cell is populated, since the order of articles is randomized, so the reverse orders are excluded.

4.2 Results

The survey ran for about one week and concluded with 11 participants. Nine participants fully answered the survey, whereas the remaining two only completed partially. The partial results were also included in the analysis. The final results are shown in table

[2]Namely NZZ am Sonntag, Tages-Anzeiger SonntagsZeitung and SonntagsBlick.

		Truth	
		Same outlet	Different outlet
Answer	Same outlet	55	53
	Different outlet	30	71
	Indifferent	1	1

Table 4.2: Confusion matrix classifying survey results between the truth and answers.

4.2. The total number of answered questions is split by the solution (or truth) and what was answered by the participants. For example, for 55 times, a question with articles from the same outlet was answered correctly, whereas for 30 times, the participants thought different news outlets were involved. In summary, out of the 211 questions asked, 126 (59.71%) were answered correctly. Nine questions were not answered due to two aborted surveys.

Using the results from the survey to conduct a two-tailed significance test, a *p-value* of 0.03896 is calculated. The p-value is under the usual threshold of $\alpha = 0.05$, which implies statistical significance. Therefore, the null hypothesis can be rejected, meaning that most likely, a discernible difference in writing styles of different news outlets exists.

4.3 Supplementary Observations

With the main question answered, the focus will be shifted on other observations. Five additional aspects of the results of the survey are examined. Comparing the results by question type, the correlation between time spent and performance, the percentage of correct answers per participant, per question and per outlet pair.

4.3.1 Correct Answers per Question Type

The questions can be categorized into two types based on its article pair: from the same outlet and from different. For questions with articles from the same outlet, 63.95% were answered correctly. 56.80% for questions with articles from different outlets. It seems that participants were better at confirming that both articles have the same style than to distinguish them.

4.3.2 Time Spent per Participant

The survey recorded the time a participant spent on answering the survey. The relationship between the time spent and the percentage of correct answers might be something worth looking into. Do participants perform better if they spend more time answering a question?

Figure 4.1 illustrates the data in a scatter plot. Calculating the Pearson correlation coefficient on the data yields a value of 0.343 alongside a p-value of 0.493. While the correlation coefficient suggests that there is a weak positive correlation between these two variables, the high p-value eliminates any statistical significance for this result.

It should also be noted that one single outlier was removed beforehand, due to having a time value of almost two days. This exceptionally large time interval can probably be attributed to the participant leaving the browser tab with the survey in the background. It is also possible that the other time values could also include time stretches where the participant was not actively engaged with the survey, but to a lesser extent.

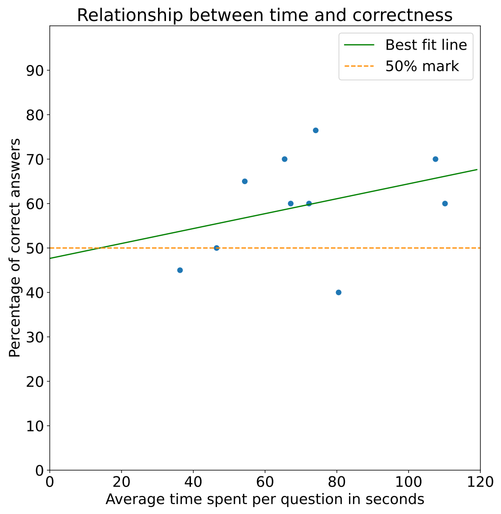

Figure 4.1: Relationship between time and percentage of correct answers

4.3.3 Correct Answers per Participant

Looking at the percentage of correct answers for each participant does not reveal any surprising insights. The average percentage lies at 60.07% with a standard deviation of 10.62%. Most participants show approximately similar performances. The data also show no outlier, meaning that it is unlikely that someone is able to consistently tell writing styles apart. Figure 4.2 displays visualizations of the data.

4.3.4 Correct Answers per Question

Another perspective to examine the results is to look at the percentage of correct answers per question. The results are illustrated in figure 4.3. With an average of 59.01% and a standard deviation of 18.87%, the results are more diverse in comparison to the perspective per participant.

Questions Q4, Q8, Q9, Q13, and Q18 have percentages over 80%, suggesting that they possess some characteristics the participants have identified. Questions Q4, Q8,

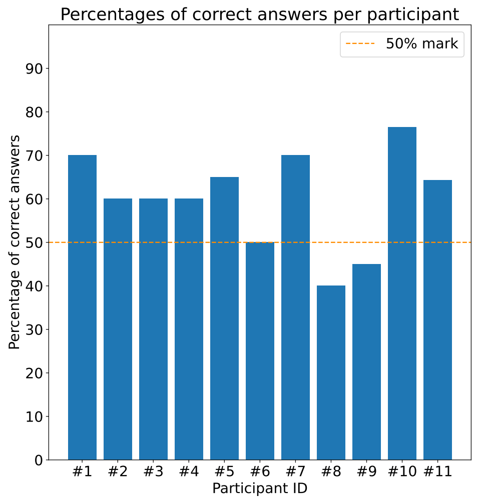

Figure 4.2: Number of correct answers per participant

Q13 and Q18 have article pairs with different sources. When comparing an article pair from these questions, the most apparent difference lies in formality. One article is written in a very neutral style, whereas the other is written more casually with more rhetorical devices and other embellishments. This gap in styling is most likely the reason they got distinguished correctly. Question Q9 is the only one out of the group with its article pair coming from the same outlet and does not exhibit the aforementioned styling difference.

A single outlier is present on the lower end with question Q14. The two articles are from separate outlets. However, they are written in a rather plain and neutral style, which is probably why so many participants did not answer correctly.

4.3.5 Correct Answers per Outlet Pair

Looking at the results from the perspective for each outlet pairing, there is an average of 59.47% of correctly answered questions with a standard deviation of 10.15%. The results are illustrated in figure 4.4. Looking at the individual results, the pairings *Blick-/Blick*, *TA/TA*, *Blick/SRF* and *Blick/TA* have the highest percentage, while *SRF/NZZ* and *SRF/TA* have a percentage under 50%. This data shows how the choice of news outlets influence the results. For example, pairings including *Blick* consistently have a percentage higher than 50%, which may signify that it possesses a more distinct writing style compared to others.

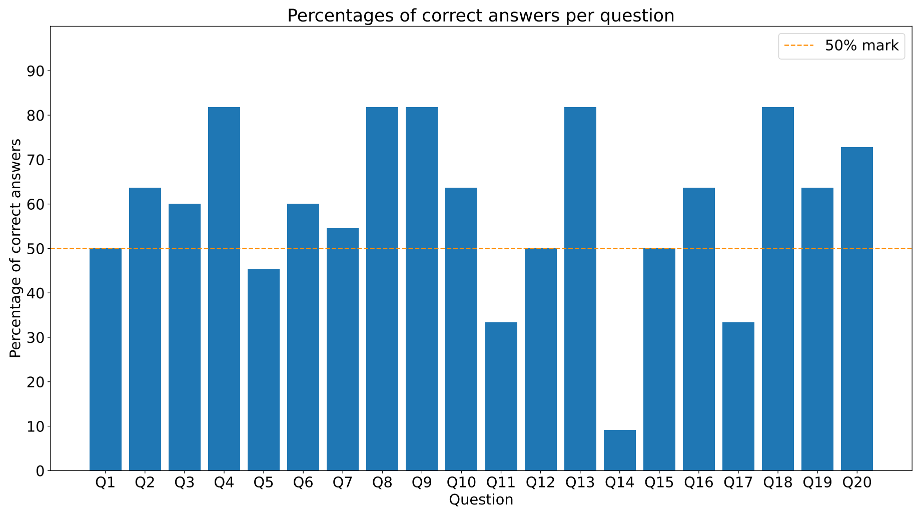

Figure 4.3: Number of correct answers per question

4.4 Interpretation

By rejecting the null hypothesis, the survey shows that readers are able to distinguish writing styles of different news outlets more often than not. Some information is present in the news articles that readers detect, which in turn allows them to tell them apart. In regard to text style transfer, these results imply that in theory, it should be possible to train a language model to pick up the differences.

However, it should be noted that the task the participants of the survey had to perform differs from the task of the style transfer model. The task of the participants of the survey solely consisted of deciding whether two articles came from the same source. In contrast, model is tasked to learn every writing style and apply them by rewriting a text with that specific style. A more suitable task for the survey that is more representative for a direct comparison would be instructing the participants to guess a specific news outlet of an article, or even rewriting entire texts. Unfortunately, such a test would require the participants to learn multiple writing styles, which exceeds the scope of this thesis by far.

Furthermore, another possible explanation on why the style transfer model produced underwhelming results may be the writing styles being too similar. While the evaluation showed how readers are indeed able to distinguish articles from separate news outlets by writing style, the rate of success is approximately 60%, which is not remarkably higher than complete random chance at 50%. The difference in writing styles might be too minuscule for the current model to pick up during training.

As for Informfully experiments where the source outlet of articles are not disclosed to the participants, researchers must carefully design to deal with issue this survey

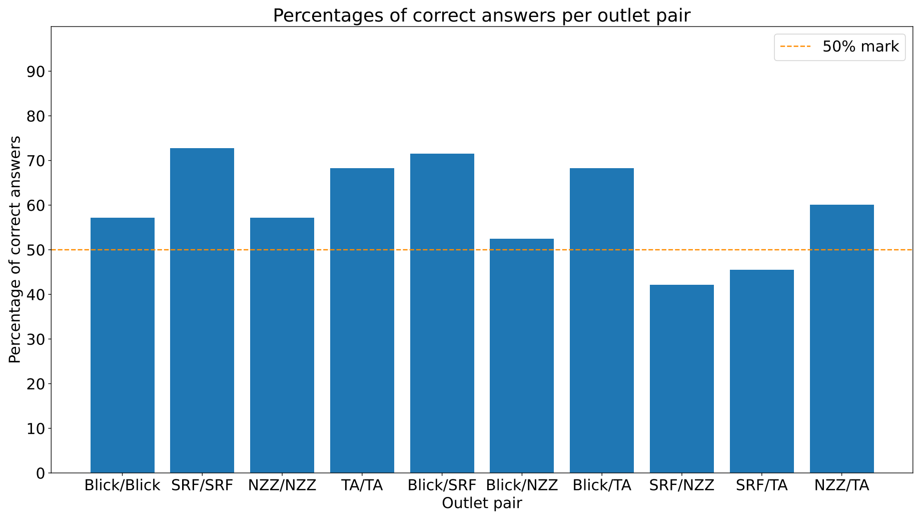

Figure 4.4: Number of correct answers per article pairing of news outlets: Schweizer Radio und Fernsehen (SRF), Blick (BLICK), Tagesanzeiger (TA) and Neue Zürcher Zeitung (NZZ).

confirmed. Special attention must be paid to ensure that participants recognizing writing styles of news outlets to not affect the results in a negative way. Although it does not seem that readers are able to consistently distinguish writing styles, it is still something to watch out for. Ideally, the text style transfer model with its current methodology would have solved this problem. An improved version has the potential to be useful in practical use cases.

5 Limitations and Future Work

This thesis was exploratory in nature. There were many aspects that were unpredictable during the planning phase. The fact that TST of writing styles is still far from being a well-researched topic contributed to the uncertainty. In the end, the product of this thesis is one with a few limitations.

The results from the survey showed that there is a difference between writing styles of news outlets that is discernible by a human reader, albeit a small one. While the current model is unable to learn and reproduce the writing styles of the selected news outlets, the methodology could be improved to obtain better performance. In this chapter, certain limitations are discussed along with possible methods to resolve them.

5.1 Poor Data Quality

What is arguably the greatest limiting factor of the trained model was the data. The data quality of articles from the archive used as training data was very poor. Despite the best efforts to clean up the data with pre-processing, it still contained many inconsistent patterns producing many unusable text lines for training. Examples include the names of authors or news agencies, image captions, hyperlinks, location and publication dates, tables showing sports results or articles that are just headlines of other articles. Further complicating the issue, each news outlet has their own format with unique problems, and may even have changed over time. A general data cleaning solution for every article would therefore not work well, so multiple custom-tailored processes would be ideal.

Instead of cleaning up the current article archive, an alternative solution to the data quality problem would be to use an entirely different dataset with a better structure. Such a dataset was unfortunately not available at the time of writing of this thesis.

The new scraper system that was created as a part of this thesis stores them in a well-structured format, leading to much higher data quality. Another benefit of the new scraper over the archive is the greatly increased number of news outlets it scrapes from. The problem with these articles, however, is that the current number of scraped articles is not nearly enough to adequately train a model, since the new scrapers only ran for a few months. Scraping older articles is also not possible without major adjustments. The new scrapers were already finished before the work on text style transfer started, so this requirement could not be implemented in a timely manner.

5.2 Increased Context Length

In the methodology of the style transfer model, the input length was already extended to include three sentences. Ideally, paragraphs or even entire articles are taken into consideration when transferring styles. Entire articles are unfortunately too large to process, being limited by the VRAM of graphics cards.

Paragraphs would be an ideal size, as sentences of a same paragraph are typically more semantically connected. The archive of news articles that was used for training had no separated paragraphs, so it was not possible to do this.

The newly acquired articles from the new scraper would be suitable to be processed in paragraphs, since it stores paragraphs separately. However, with the aforementioned issue with having too few articles at the time, it could not be implemented.

5.3 Additional Noising Methods

The BART paper [Lewis et al., 2019] describes more noising methods. With the current model, only token masking and token deletion were implemented. Token masking is described as crucial for any task, whereas token deletion outperforms token masking for generation task. The remaining three methods are sentence permutation, document rotation and text infilling. With sentence permutation, every sentence in a sample is randomly shuffled, tasking the model to rearrange the sentence to its original form. Document rotation randomly chooses a token and starts the sample at that position and the actual start is appended at the end. The model is then tasked to learn to recognize the beginning of the sentence. Text infilling replaces entire spans of tokens with a single mask token, which is basically a more aggressive version of token masking.

The additional methods were not implemented because there are no apparent benefits in regards of style transfer. Implementation and a subsequent evaluation of these methods would still be interesting to test out.

6 Conclusion

The idea of this thesis started as a response to an issue with the Informfully system. Concerns were raised on how participants of an experiment might recognize a writing style of a certain news outlet while reading an article. If the experiment requires participants to not know the article source, it could lead to skewed results whenever a bias against a news outlet is present.

In an attempt to resolve this issue, multiple approaches were thought out. The first idea was to utilize machine learning to simplify every article to the same complexity level, in hopes of erasing any characteristics that may reveal the source. However, insufficient parallel data for training and other problems made this method impractical for the use case. Instead, an alternative approach was explored: text style transfer. A deep learning model is trained to learn and transform texts into another style, while preserving the original content. The issue situation regarding availability of parallel data is even worse with no data at all, but can be circumvented with a denoising autoencoder or masked-language modeling, only requiring a monolingual dataset. Noise is intentionally introduced into article texts and then given to the model, which in turn tries to restore the text to its initial state. In theory, the model learns the various aspects of a specific writing styles with this method. A trained model can then be used to normalize articles to be used in experiments.

Unfortunately, results were rather disappointing. Any texts given to the model were mostly unchanged after processing, regardless of which style was requested. Despite attempts to improve the results, only minor differences were made between the input and output texts that did not alter the style or content at all.

Rather than spending the remaining resources on improving the methodology, a survey was set up to test human performance on detecting text differences. There may be no between writing styles of news outlets for the model to pick up at all, considering the output text was not different from the input. In the survey, participants are shown randomized pairs of articles and are then tasked to guess if the articles come from the same news outlet or not. In the end, results from the survey show that readers are able to answer correctly with a 60% accuracy.

In conclusion, while the text style transfer model could not successfully be trained to rewrite articles into a style of another news outlet, several lessons can be learned from the survey. The result from the user study is a reminder to pay attention when designing experiments dealing with news articles. Participants might pick up more information that was intended solely by recognizing writing styles. To deal with this issue, the current text style transfer model could be further developed, implementing the suggestions to improve it a point where it can be used to normalize articles and many other use cases one can think of.

References

[Bahdanau et al., 2015] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

[Beltagy et al., 2020] Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *CoRR*, abs/2004.05150.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

[Clark et al., 2020] Clark, K., Luong, M., Le, Q. V., and Manning, C. D. (2020). ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

[Gage, 1994] Gage, P. (1994). A new algorithm for data compression. *C Users Journal*, 12(2):23–38.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

[Jin et al., 2020] Jin, D., Jin, Z., Hu, Z., Vechtomova, O., and Mihalcea, R. (2020). Deep learning for text style transfer: A survey. *CoRR*, abs/2011.00416.

[Lan et al., 2019] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942.

[Lewis et al., 2019] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

[Liu, 2019] Liu, Y. (2019). Fine-tune BERT for extractive summarization. *CoRR*, abs/1903.10318.

[Liu et al., 2020] Liu, Y., Gu, J., Goyal, N., Li, X., Edunov, S., Ghazvininejad, M., Lewis, M., and Zettlemoyer, L. (2020). Multilingual denoising pre-training for neural machine translation. *CoRR*, abs/2001.08210.

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

[Radford and Narasimhan, 2018] Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training.

[Sanh et al., 2019] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

[Sennrich et al., 2016] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

[Syed et al., 2019] Syed, B., Verma, G., Srinivasan, B. V., Natarajan, A., and Varma, V. (2019). Adapting language models for non-parallel author-stylized rewriting. *CoRR*, abs/1909.09962.

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

A

Appendix

A.1 Scraper Pipeline

The process of getting articles from the news outlets and saving them into the database was not particularly well structured in the past version of Informfully. The scrapers had to be adjusted repeatedly, because the news outlets often changed the format of how they presented their articles. The uncertainty of further changes in the future as well as the need of timeliness of the news articles meant that any updates to the scrapers were rushed.

The scrapers were separate Python scripts that were each individually executed by cronjobs. For a new scraper, the majority of the code was copied over to the new one, only making some smaller changes to adapt to the new outlet. This practice led to a lot of duplicated code. For example, the functionality to insert the articles into the database was virtually the same for every scraper.

The new scraper pipeline architecture aims to reorganize the scraping process, ultimately making it simple and quick to become familiar with the code as well as to implement or change any features. An overview of the new system is illustrated in figure A.1. The installation process only requires Python 3 and modules listed in **requirements.txt**.

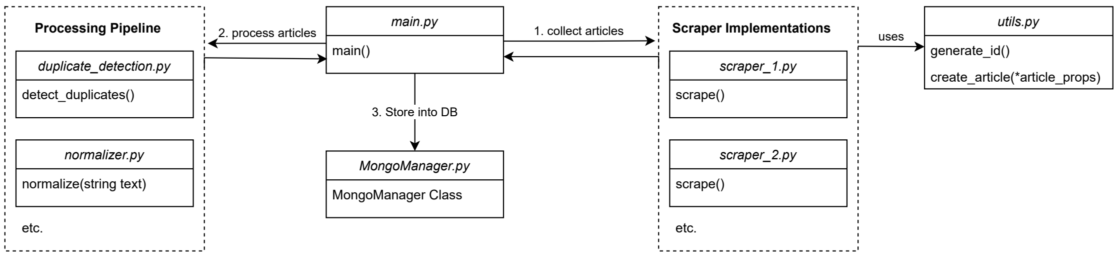

Figure A.1: Scraper architecture of the Informfully back end.

Description	**Variable names**
MongoDB config	MONGOHOST, MONGOPORT
DB credentials	DBUSER, DBPASS, DBNAME
SSH config	USESSH, SSHHOST, SSHUSER, SSHPASS
Mail config	MAILONERROR, MAILFROM, MAILTO
SMTP config	SMTPUSER, SMTPPASS, SMTPHOST, SMTPPORT
Geckodriver config	GECKODRIVERPATH, FIREFOXPROFILEPATH

Table A.1: Environment variables used in scraper system

A.1.1 Scraper Package Structure

The files for the individual scrapers and files for shared utilities, for example the code for database access, have been collected into a single python package. It is the intention that the modules will be imported and used by a *main* python script from outside the package.

Inside the scraper package, there are two folders. The *scrapers* folder contain the scraper code, a file for each outlet. The *pipeline* folder contain the files for the processes after the articles have been scraped and before they are finally inserted into the database. New code in form of new scrapers or processing steps can be arbitrarily inserted into those folders.

A.1.2 Environment Management

A configuration management system was added to the pipeline. This allows variables, which differ depending on the specific deployment, to be stored and changed in a single location. And unlike previously, credentials the database connection or for news outlet accounts are not stored inside the code files, but rather in a separate **.env** file that does not get checked into the code repository for increased security.

Specifically, the *dotenv*[1] was chosen to accomplish this job. While it is usually intended for *Node.js* projects, the very basic syntax as well as available implementations in pretty much every common programming languages made it a well suited choice.

To load the environment variables, the *main.py* program calls the load_env() function from the dotenv module. This will set the environment variables on a OS level. From within python code, the variables can be accessed via os.getenv().

A list of environment variables is shown in table A.1.

A.1.3 MongoDB Manager

The database system often ran on a different machine than the one where the scraper pipeline was. The scraper pipeline were usually executed on personal machines of researchers, whereas the database that contained the relevant data resided on a central

[1]Python package for dotenv *https://pypi.org/project/python-dotenv/*

server. There were also cases were the researchers were not adept at installing a MongoDB instance on their personal computers. For these reasons, a manager class has been written to simplify the connection to the Mongo database.

The MongoDB manager module packs the logic of establishing the SSH tunnel as well as the connection to the MongoDB instance into a class. It retrieves the required credentials from environment variables and automatically returns a *MongoClient*[2] object. The class is implemented as context managers, so it can be used with the *with* keyword. Once the code reaches the end of the code block or in case of an error, the SSH tunnel will be closed automatically.

A.1.4 Logging

A logging module has been created to log any meaningful event happening during the scraping. Every single log will contain a short message, the location where it was logged, a timestamp and an id which uniquely identifies a single execution run of the scrapers. This logging tool is particularly useful for troubleshooting errors.

Any abnormalities in the scraper pipeline should be handled in a timely manner. Especially during an ongoing experiment, it is vital that the flow of new articles is not halted. In the event of a fatal error in the pipeline, the administrator is notified immediately via email with all relevant information. With the *run id*, other logs belonging to the same run can be queried from the database to get a timeline of events leading up to the error.

A.1.5 Scrapers

The individual scraper python modules are each required to implement a function named **scrape()**. The main program will import the module and try to call it, expecting it to return a list of article objects. The actual implementation of this function is not enforced, for example by an abstract class, as it would make the code more convoluted for something as simple as one function. Functionalities that are shared among the scrapers are stored in a separate *utils* file. For instance a function to create an article object, ensuring uniformity in object field naming and default values.

Despite the goal of the scrapers being the same, i.e., to obtain content from news articles, the sheer variety of different formats, types of sources of information and paywalls from news outlets rendered the task of implementation very challenging. Ultimately, a rough pattern has surfaced for every outlet consisting of two parts. First, get a list of the most recent articles with either URLs or an identifier for the API. Afterwards, iterate through the list and obtain the necessary information by scraping the HTML page or directly accessing the API if possible.

Nowadays, RSS feeds are ubiquitous on any news website. The feeds can be used to receive a list of news articles in formatted as an XML file. The scraper gets this list

[2]Official documentation of Pymongo: *https://pymongo.readthedocs.io/en/stable/*

and parses them with the *feedparser*[3] module. At this stage, some useful information can already be extracted from the feed such as the URL, the title and a description. However, the article content, the text itself, does not come with the feed. The article text for each article is taken from an API. In cases where an API is not available, the content is scraped from the HTML page of the URL. To obtain the HTML code from the URL, the built-in python package *urllib*[4] is used. To simplify the process of extracting the relevant information out of an HTML page, the *Beautiful Soup*[5] is used and can help parse the HTML content into an organized data tree with built-in methods to navigate, search and modify.

With every property of an article at hand, a proper object can be constructed using the utility function save_article(). This function makes sure that all the necessary fields are initialized with default values if the data could not be obtained using the scraper.

Within the small selection of scrapers, there are a few that do not exactly follow the pattern *RSS feed → HTML scraper* or *RSS feed → API*. For example, *Weltwoche* does not offer an RSS feed. In order to get a list of the newest articles, another page can be visited showing an overview of the newest edition. However, the actual content of interest in the page is not directly included in the initial HTML data. The article elements are dynamically loaded in later on using JavaScript. To automate this process, Selenium[6] along with Geckodriver[7] is used to open a browser, navigate to the URL and execute the javascript to get a finalized page, from which the article list can be extracted using Beautiful Soup. The scraping of individual article functions the same as other HTML based scrapers. Naturally, this kind of scraping feeds takes more time compared to RSS feeds since a lot more processing is required to render a dynamic webpage.

A.1.6 Processing Pipeline

After every article have been scraped and are present in form of a single list, the system will not directly persist them permanently into the database yet. There are a few processing steps necessary to ensure that the articles are prepared to a state that is optimized for the purposes of Informfully.

In short, the processing of articles consists of two steps:

1. Duplicate detection
2. Normalization

Duplicate Detection

News outlets often take pre-written news articles from news agencies and publish them on their own websites. When multiple news outlets take the same article, both instances

[3]Code repository of feedparser: *https://github.com/kurtmckee/feedparser*
[4]Documentation of urllib: *https://docs.python.org/3/library/urllib.html*
[5]Documentation of BeautifulSoup: *https://www.crummy.com/software/BeautifulSoup/*
[6]Selenium website: *https://www.selenium.dev/*
[7]Code repository of Geckodriver: *https://github.com/mozilla/geckodriver*

will be scraped and put into the database. Both articles will be shown in the app and because the name of the news outlet source is omitted, the user would see two virtually identical articles. This is likely to be very confusing for the participant, so an additional step had to be present in the pipeline to detect and handle duplicate articles.

However, news outlets usually make some minor changes to the original article content to fit their own format, therefore a plain equality check on the entire article text will not work. In the new implementation, the two articles are first split into n-grams, sequences of words from the the article text of a specific length. Afterwards, a percentage is calculated of how many sentences are shared between the two articles. If the overlap percentage is over a certain threshold, the two articles will be considered as duplicates.

Every new article from the scrapers will undergo this step. The articles are compared against each other and also from the existing articles in the database pairwise. Due to the time relevancy of news articles, duplicates typically do not have a large time span between them, so the selection of articles from the database are limited by a time window of a few days.

Text Normalization

Especially when directly scraped from the websites, news articles may contain idiosyncrasies such as spelling variants of words, formatting (numbers, dates, and headlines) as well as self-references (names of the publication). These anomalies may indicate the source of the article, which is undesirable. The goal of the normalization step is to remove them to ensure a uniform presentation for the app.

The normalization process is implemented as a list of tuples. The first element in the tuple is a regular expression to detect the abnormalities, with the second element being the string of text that serves as the replacement. For example, the term *Tagesanzeiger* will be replaced with a generic, non-descript *[Zeitung]*. While the capabilities of regular expressions are barely utilized at the moment, the functionality will be preserved in case there will be problems in the future that will require more complex patterns.

A.1.7 Deployment

Deployment of the scraper system is a straightforward process. To use it on a machine, simply clone the files from the *scraperpipeline* folder of the code repository using git. The scraper pipeline needs a recent version of Python (>=3.6). In addition, a number of python packages listed in the file *requirements.txt* must be installed beforehand. This can be done using the pip package manager. The use of a virtual environment is highly advised, for example with Python's built-in *venv* module or with *conda*[8].

Running in parallel to the scrapers, a MongoDB instance is used to persist the articles. The same database instance can also used by the *Informfully* backend and app.

Executing the scrapers can be accomplished by calling the *main.py* python script. It will automatically collect the news articles and put them into the database. Which

[8]Documentation of Conda: *https://docs.conda.io/en/latest/*

scraper are used can be modified by changing the list in the main file. To avoid having to start the scrapers manually, setting a cron job to periodically call the main file is highly advised.

A.1.8 Limitations

One issue that is almost certainly going to come up after deployment are changes to APIs and article formats. It is less likely to happen to scrapers relying on APIs, since the more frequent layout changes do not affect APIs. It is also highly dependent if the news outlet will send out notifications on any updates to their websites, preferably with enough time, to update the scraper. For any unannounced changes, the logging system should catch any errors during scraping and send out an email to the admin if it is set up.

Even during development, one news outlet changed their website layout, completely breaking the scraper in the process. Fortunately, due to the modular nature of the new architecture, the issue could be quickly resolved.

List of Figures

List of Tables