Proceedings of the Second Workshop on Language Technologies for Historical and Ancient Languages (LT4HALA 2022), pages 1–9
Language Resources and Evaluation Conference (LREC 2022), Marseille, 25 June 2022

Identifying Cleartext in Historical Ciphers

Maria-Elena Gambardella, Beáta Megyesi, Eva Pettersson
Dept. of Linguistics and Philology, Uppsala University
mariaelena.gambardella@gmail.com, {beata.megyesi, eva.pettersson@lingfil.uu.se}

Abstract

In historical encrypted sources we can find encrypted text sequences, also called *ciphertext*, as well as non-encrypted cleartexts written in a known language. While most of the cryptanalysis focuses on the decryption of ciphertext, cleartext is often overlooked although it can give us important clues about the historical interpretation and contextualisation of the manuscript. In this paper, we investigate to what extent we can automatically distinguish cleartext from ciphertext in historical ciphers and to what extent we are able to identify its language. The problem is challenging as cleartext sequences in ciphers are often short, up to a few words, in different languages due to historical code-switching. To identify the sequences and the language(s), we chose a rule-based approach and run 7 different models using historical language models on various ciphertexts.

1. Introduction

Since humankind created written language there has been a need to send messages to each other in a safe way, without the interference of a third party.

Historical ciphers are encoded, hand-written manuscripts aiming at hiding the content of the message. Historical ciphers usually contain encoded sequences of various symbols, so called ciphertexts, as well as cleartexts, i.e. non-encrypted text written in a known language. All text sequences that have not been encrypted, but are left in its original form are called cleartext.

During the decryption process, the ability to distinguish cleartext from ciphertext is essential, since cleartext can give clues to the underlying language of the cipher and help us in the historical interpretation and contextualisation of the manuscript. By analyzing the cleartext of the cipher we can make educated guesses about the topic and the context of the document, which can lead to the decryption of important keywords, or the encoded named entities, such as locations or names of persons (Megyesi et al., 2019).

Cleartext might be a longer text, or short sequences of words making language identification more challenging. The scribe might use one or several languages in the same cipher, as code-switching was common in our history. And while ciphertexts are often represented by a specific symbol system designed for the particular cipher, such as digits, alphabets, graphic signs or a combination of them, cleartext consists of the alphabet of the language(s) involved. An example of cleartext and ciphertext sequences following each other in a historical cipher from 1625 is illustrated in Figure 1.

The goal of our study is to automatically identify the cleartext sequences in ciphers, and their language(s). We use historical language corpora for which we create word-based and character-based language models of various orders from unigrams to fivegrams. We build models for 16 European languages: Czech, Dutch, English, French, German, Greek, Hungarian, Icelandic,

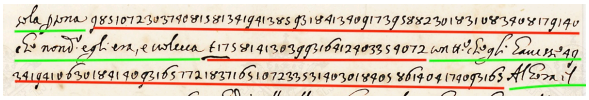

Figure 1: Excerpt of a cipher with ciphertext (in red) and cleartext (in green). Record 69 in the DECODE database (ASV, 2016b).

Italian, Latin, Polish, Portuguese, Russian, Slovene, Spanish and Swedish.

The work has been carried out within the DECRYPT project (Megyesi et al., 2020) aiming at the development of a research infrastructure for the study of historical cryptology. More specifically, the purpose of the project is to create resources and tools for (semi-)automatic transcription, cryptanalysis and decryption of historical encrypted documents.

In the remaining part of the paper, we give an overview of language identification in historical text followed by attempts made with regard to language identification in ciphers. In Section 3, we present the method to automatically segment cleartext and ciphertext in ciphers, and identify the language of the cleartext. In Section 4, we describe the results and in Section 5, we conclude our findings.

2. Background

Automatic language identification of a text is claimed to be a solved problem in natural language processing. When we browse or translate text using Google, the system identifies the language of the text with high accuracy. This applies especially to longer and modern text. However, when only a few words are typed in and/or when we are dealing with historical text, identifying the language becomes harder with less reliable results. In this section, we give an overview of language identification in general, and then we describe previous studies on attempts made for language identification in historical ciphers in particular.

2.1. Language Identification

Language identification is the task of recognizing the language a text is written in. The aim is to create systems that are able to recognize any human language, being it in the form of speech, sign language or handwritten text. The methods used are many, ranging from decision rules to neural networks, and the task can be applied to many areas. In the field of translation the use of language identification can be dated back to the 80s when (Beesley, 1988) created a prototype system for language identification for online texts. Another use of language identification we can find is in multilingual document storage and retrieval where one of the challenges is to disambiguate the so called "false friends", i.e. a word that holds different meanings in two languages, but is written in the same way in both languages (e.g. *gift* meaning "present" in English, but "married" in Swedish).

Texts in which several languages are present and also alternated, a phenomenon called code-switching, are commonly occurring, both in modern and in historical texts. Within the NLP community, several studies have been carried out to identify where code-switching occurs and which languages are involved. The First and Second Workshops on Computational Approaches to Code Switching organized in 2014 (Diab et al., 2014) and 2016 (Diab et al., 2016) were the first workshops dedicated to the topic. They organized shared tasks to identify languages in code-switched data. The most popular and successful approaches were based on machine learning algorithms. In (Shirvani et al., 2016), they performed token-level identification using a Logistic Regression model with L2-regularization to generate language labels on the tokens. The results outperformed the other participants, ranking the system at first place for the language pair Spanish-English.

There have been attempts in using deep learning algorithms to perform code-switching identification, with very good results. In (Samih et al., 2016), the authors present a long short-term memory (LSTM) approach relying on word and character representations, where the output is fine-tuned using a conditional random field (CRF) classifier to capture contextual meaning. They did not use any linguistic resources, making the model language independent. The results outperformed the other participants ranking the system at first place for the language pair Modern Standard Arabic-Dialectal Arabic and second for the language pair Spanish-English at the Second workshop.

Despite the highly rising interest in the use of deep learning ones, there are still researchers interested in using rule-based methods. In (Chanda et al., 2016), the authors tag their dataset of Spanish-English tweets at a word level and use three different dictionaries to recognize the language of each word. If the word is tagged as both languages, it is given to a Predictor-Corrector algorithm, which checks the tag given to the previous and next word; if they are the same it will give the same tag to the mixed word, otherwise it will tag it as ambiguous. Although the results achieved in the second workshop in 2016 do not outperform the other participants' systems, they outperformed the baseline.

2.2. Language Identification in Ciphers

When it comes to language identification in ciphers, like research on detecting cleartext in a cipher and identifying its language, this task presents a noticeable lack of literature. To the best of our knowledge, the only attempt in language identification in this field has been carried out in (Pettersson and Megyesi, 2019), where the authors present an approach to automatically mapping ciphertext sequences to keys in order to return the plaintext from the ciphertext by using homophonic substitution. Historical language models are consulted to guess the language used to write the decrypted plaintext. They use three ciphertexts from the DECODE database (Megyesi et al., 2019) for training and one for evaluation.

The first step for the cipher-key mapping algorithm consists of storing code-value pairs and the length of the longest code processed from the key file. In the second step, the transcribed text is matched against the code-value pairs. The search method is a non-greedy search-and-replace mechanism, which consists in checking the length of each word with the longest code in the cipher. Different approaches to matching are applied depending on the length of the given word: 1) if it is shorter than or equal to the longest word, it is checked whether the word can be matched with a code and if so, the word is replaced with the value attached to that code; 2) if the word cannot be matched with a code or if its length is longer than the longest word, then the algorithm iterates over the word, character by character, and try to match these characters with a code: if the approach is successful, the current character is merged with the succeeding character, and the algorithm tries to match the longer sequence with a code until its length is equal to the longest word's length. If there is no match when the word reaches the longest word's length, the sequence is replaced by a question mark. The third step is to identify the language of the decrypted text that was generated in the previous steps. This is done based on word-based language models from the HistCorp webpage,[1] where the plaintext words are compared to the words in the language model for each language. The model outputs a ranked list of these languages showing the percentage of words in the plaintext file that are found in the model for each language.

Next we turn to the description of our work.

3. Method

In this section, we will describe our approach to detect cleartext and identify its language. We start by describing the data, both the ciphers used and the historical corpora for the creation of the language models.

[1] https://cl.lingfil.uu.se/histcorp/langmodels.html

3.1. Ciphers and Transcriptions

To detect cleartext in ciphers, we need transcribed manuscripts with ciphertext and cleartext sequences marked along with their language ID. The DECODE database[2] contains a collection of almost 3000 ciphertexts and keys from Early Modern times in Europe (Megyesi et al., 2019). Over 400 ciphers are available with their transcriptions. All transcribed manuscripts follow the same guideline for consistency (Megyesi, 2020). An example of an original cipher with cleartext followed by ciphertext is exemplified in Figure 2 and its corresponding transcription is shown in Figure 3.

First, the transcription begins with comment lines (starting with "#") which provide information about the file. Then, the content of the cipher is transcribed, symbol by symbol and row by row. Digits are transcribed as numerals in ASCII (1 is transcribed as "1", 2 as "2", 0 as "0"), along with the Latin alphabet including capitalized letters (a is transcribed as "a", capital B as "B"), and punctuation marks (".","!",). For other symbols, we use the Unicode names. Each symbol is transcribed separately, and we add a space between each symbol. In case of spacing in the original, multiple spaces are introduced as these might mark word boundaries in the underlying plaintext. Uncertain symbols are transcribed with added question mark "?" immediately following the uncertain element. To be able to distinguish between ciphertext and cleartext, cleartext sequences are marked in brackets as:

< CLEARTEXT LANG Symbol sequence >.

If the manuscript contains several lines of cleartext, each new line is represented by a new CLEARTEXT tag. LANG denotes the language the cleartext is written in, marked by a language ID as defined by ISO 639-12 two-letter codes for languages (e.g. ES for Spanish). If there is some doubt about the cleartext language, the language ID is defined as unidentified (UN).

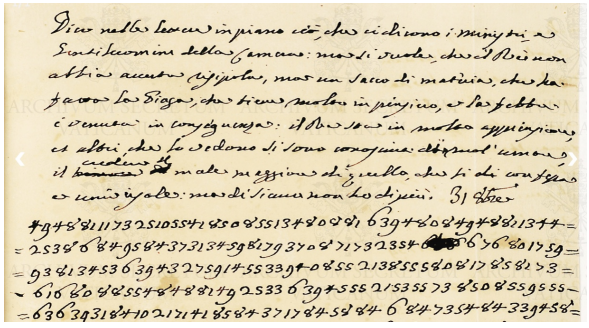

Figure 2: Excerpt of a cipher with ciphertext and cleartext. Record 198 in the DECODE database (ASV, 2016a).

The transcriptions were preprocessed to make computation easier. We removed all question marks representing uncertainty (e.g. 8? is returned as 8) and kept the first alternative in case of multiple interpretations (e.g. 6/8? is returned as 6). We then converted all Unicode names into symbols (by using the `lookup` function in the `unicodedata` module). We removed all spaces between the codes as well as the cleartext and plaintext tags. Finally, we corrected remaining errors caused by the manual transcritpion, such as missing brackets and wrong unicode names. The result is a collection of texts which looked exactly as their original historical manuscripts without any annotation.

[2] `https://de-crypt.org/decode`

```
#CATALOG NAME: Segr. di Stato/Portogallo 117/5
#IMAGE NAME: 104r.jpg
#TRANSCRIBER NAME: Midas
#DATE OF TRANSCRIPTION: May 5, 2017
#TRANSCRIPTION TIME: 20 min.

#IMAGE NAME: 104r.jpg
<CLEARTEXT IT Dice nella lettera in piano ciò che ci dicono i ministri, e>
<CLEARTEXT IT Gentiluomini della Camera: ma si vuole che il Re non abbia>
<CLEARTEXT IT avuto risipola,  ma un sacco di materia che ha>
<CLEARTEXT IT fatto La Piaga, che tiene molto in pensiero, e la febbra>
<CLEARTEXT IT è venuta in conseguenza: il Re sta in molta apprensione,>
<CLEARTEXT IT et altri che lo vedono si sono conosciuti ???? umore,>
<CLEARTEXT IT il credere il male peggiore di quello che si confessa>
<CLEARTEXT IT è universale: ma di sicuro non ho di più. 31 8bre>
4 9 4 8 8 1 1 1 7 3 2 5 1 0 5 5 4 1 8 5 0 8 5 5 1 3 4 8 0 8 8 1 6 3 9 4 8 0 8 4 9 4 8 8 1 1 3 4 4 =
= 2 5 3 8 6 8 4 9 5 8 4 3 7 3 1 3 4 5 9 8 1 7 9 3 7 0 8 7 1 7 3 2 3 5 4 67 6 7 6 8 0 1 7 5 9 =
= 9 3 8 1 3 4 5 3 6 3 9 4 3 2 7 5 9 1 4 5 5 3 3 9 4 0 8 5 5 2 1 3 8 5 5 5 8 0 8 1 7 8 5 8 1 7 3 =
- 6 1 6 8 0 8 8 5 5 4 8 4 8 8 1 4 9 2 5 3 3 6 3 9 4 5 5 5 2 1 5 3 5 5 7 3 8 5 0 8 5 5 9 5 5 5 -
= 6 3 6 3 9 3 1 8 4 1 0 2 1 7 1 4 1 8 5 8 4 3 7 1 7 8 4 5 8 8 4 6 8 4 7 3 5 4 8 4 3 3 9 4 5 8 =
```

Figure 3: Excerpt of a cipher transcription with ciphertext and cleartext. Record 198 in the DECODE database (ASV, 2016a).

The dataset used in our study consists of 214 documents in 8 different languages. Transcriptions of two longer enciphered manuscripts are also present: the Borg (Aldarrab et al., 2017) and the Copiale(Cop, 2011 2020) ciphers, dated back to the 18th and the 17th centuries, respectively. To test if the models overgenerate we also included some texts without any cleartext.

As we can see in Table 1, the language with the most documents is Hungarian, followed by French, Italian, Spanish, Latin. In the sample, we can also find documents with multiple languages such as a combination of Latin and French, where the cleartext was written in Latin and the ciphertext was decoded into French. Some languages occur in one document only, such as Dutch and Portuguese. We could not create a balanced sample, we took simply what we could get.

In the manuscripts, we find a large variation of symbols used to encode the text. In Table 2, we show the distribution of symbols across the training and test sets. The most frequent symbol set used in the ciphers are digits, representing around 78% of our data. The second most used symbol set is a combination of digits and Latin letters (around 15% of the dataset), followed by a combination of digits, Latin letters and graphic signs (around 6% of the dataset). The least used symbol set is the combination of graphic signs and Latin letters representing around 1% of the dataset.

The dataset was partitioned into 60% for training and 40% for test, with no development set. The motivation behind this choice is that we were not planning to use machine learning, so the training set does not necessarily need to have a lot more data compared to the test set and the same applies for the absence of a development set. The transcriptions of the long ciphers, the Borg

Language	Num of doc for training set	Num of doc for test set
Hungarian	32	22
French	29	20
Italian	29	20
Spanish	25	17
Latin	5	3
Latin/French	4	2
Portuguese	0	1
Dutch	0	1
Unknown	0	1
No cleartext	1	2

Table 1: Language distribution in the dataset.

and the Copiale, both with non-standard symbols, were divided into 50% for training and 50% for testing.

3.2. Language Models

Inspired by the work of (Pettersson and Megyesi, 2019), we decided to include 16 European languages, all with freely available historical corpora through the HistCorp platform (Pettersson and Megyesi, 2018). The included languages are: Czech, Dutch, English, French, German, Greek, Hungarian, Icelandic, Italian, Latin, Polish, Portuguese, Russian, Slovene, Spanish and Swedish. Historical corpora with diplomatic editions are available for all, along with pre-trained language models which are perfectly suitable and adaptable for our purposes; both for cleartext detection and language identification.

The language models are built using the IRSTLM open source toolkit (Federico et al., 2008). Every language has word-based models, including up to 3-grams, and character-based models, including up to 5-grams. The models are text files with the token in the first column and their absolute frequency in the second column. In order to use these models for both language identification and cleartext detection, we created two dictionaries: one collecting all items in the word-based language models and one collecting all items in the character-based language models (see Table 3 for size of each language model).

The motivation behind using historical language models rather than larger modern models is related to the nature of the texts analyzed. Because historical ciphers contain historical language with different spelling and vocabulary used at the time, language models built on historical texts seemed to be appropriate for our purposes.

In order to use the language models, we build two dictionaries: one collecting word-based language models and one collecting character-based language models. The motivation behind choosing a dictionary as our data structure is because of the relative speed with which items can be retrieved. Being a hash table, when we search for an item we look directly at the "slot" that holds the name of the item we are looking for and retrieve its value. This search is equal to *O(1)*, meaning that the size of the dictionary has no effect on the search, since it is constant (Miller and Ranum, 2006).

For the word-based dictionary, unigrams and bigrams were used (3-grams turned out to be computationally too heavy to be useful for our task). For the character-based dictionary, 3-grams, 4-grams and 5-grams were used. The motivation for not using unigrams and bigrams in the character-based setting, is that these short segments are more likely to be part of several different language models. For example, *ia* can be a common suffix in both Spanish and Italian, but for longer *n*-grams we can get more unique combinations for certain languages. The dictionaries have words or characters as keys and a list of tuples in the form `(language, frequency)` ranked by the second item with the first one being the one with the highest relative frequency as values.

3.3. Cleartext Detection

To distinguish cleartext sequences from ciphertexts, we were inspired by the the work of (Chanda et al., 2016), as explained in Section 2.1. In particular, we were interested to see how the approach of analyzing modern social media (Twitter) data on word level could be applied to and how well it could perform on historical texts.

We decided to experiment with various types of models. For our baseline model we chose unigrams only on a word-based level. In addition to the baseline, we tried six different n-gram combinations:

1) For the first model (Word 1gram Threshold on FRequency, W1_TFR), we considered only unigrams, but not the least frequent ones. We set a threshold of 1 on the absolute frequency of all unigrams when creating the word-based dictionary from the language models. The motivation for this is to see how removing the least frequent words could affect the results of cleartext detection.

2) For the second model (Word 1gram Threshold on Letters, W1_TL), we also considered unigrams, but only those which presented letters. In order to achieve this goal, we set a threshold where unigrams which presented digits were not considered (e.g. '23gf' or '65.'). The motivation behind this is that words containing numbers seem to be less likely to be text than code and therefore should possibly be ignored.

3) For the third model (Word 1gram Threshold on Letters + Word 2gram, W1_TL + W2), we considered a combination of bigrams and unigrams with the same threshold as in the second model (1L). The model will first check if the bigram is present in the dictionary and if not, it will split the bigram into unigrams and check if each of these is present in the dictionary. The motivation behind this combination is the fact that

Set	Digits	Graphic Signs + Latin letters	Digits + Latin letters	Digits + Latin letters + Graphic Signs
Training	96	2	20	7
Test	70	2	8	9

Table 2: Symbol set distribution in the dataset.

Language	Words	Chars
Czech	4,364,685	25,359,937
Dutch	14,549,599	87,206,041
English	90,983,314	451,860,888
French	366,437	1,964,634
German	256,039,161	1,748,530,003
Greek	11,179,688	135,546,052
Hungarian	2,169,442	13,567,260
Icelandic	983,517	5,478,104
Italian	7,635,969	48,277,101
Latin	95,181,455	663,451,162
Polish	3,203,330	16,980,174
Portuguese	3,178,447	17,342,279
Russian	25,822	283,024
Slovene	18,081,602	92,197,951
Spanish	7,381,647	41,052,708
Swedish	17,606,410	104,409,451

Table 3: Size of language models.

some words can be difficult to identify when taken individually, but when we consider their neighbouring word the process could be easier. For example, in the case of dates, such as *August 1697*, it could be easier to identify *1697* as part of a date if it is considered together with *August* than if it would be considered alone.

4) For the fourth model (Word 1gram Threshold on Letters + Word 2grams + CHaracters 2-, 3-, 4-grams, W1_TL + W2 + CH345), we considered a combination of bigrams and unigrams from the third model and added 3-grams, 4-grams and 5-grams on a character level. The model will first check if the bigram is present in the dictionary and if not, it will split the bigram into unigrams and check if each of these is present in the dictionary. If the unigram is not in the dictionary, the model will check the combination of characters in the character-based dictionary. The motivation behind adding characters to the model is the fact that in the past, words could be spelled in different ways and using character-based language models could help us capture these words, even if the word as a whole is not recognised in a dictionary.

5) For the fifth model (Word 1gram Threshold on Letters + Word 2grams + CHaracters 2-, 3-, 4-grams Threshold on Letters, W1_TL + W2 + CH345_TL), we considered a combination of bigrams, unigrams and characters as in the fourth model, but we also added the same threshold that we have on the unigrams to the characters, that is *n*-grams which presented digits were not considered (e.g. '23gf' or '65.' are not considered).

6) For the sixth model (Word 1gram Threshold on Letters + Word 2grams Threshold on Letters + CHaracters 2-, 3-, 4-grams Threshold on Letters, W1_TL + W2_TL + CH345_TL), we considered the same combination of bigrams, unigrams and characters as in the fifth model, but we also added a threshold to the bigrams, where bigrams which presented only digits were ignored (e.g. '23 45' is not considered, but '23 Agoust' is). The motivation behind having such a threshold for the bigrams is to increase our chances to capture text. Our intuition is that a combination of two numbers is more likely to be code than a combination of a number and a word and should therefore be ignored.

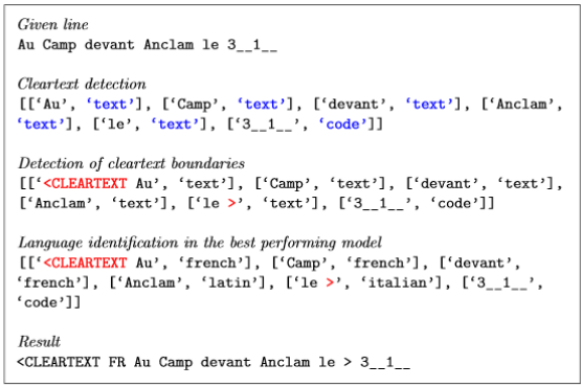

Figure 4: Example of how the algorithm works with models using unigrams.

Our algorithm analyzes each file in our data set line by line, as they are transcribed. For the baseline model and models 1 and 2, it splits each line into unigrams and we forward each unigram to a function that assigns a 'text' tag if the word is found in the word-based dictionary or a 'code' tag in the event that the *n*-gram is not found. For model 3 to model 6, we split each line in bigrams and we search for these *n*-grams in a slightly different manner: we first search for the bigram in the word-based dictionary, and if the bigram is not found we split it into its unigrams and search for each of them in the word-based dictionary again. If the unigram is not found, we search for the combination of characters: if the unigram is shorter than or equal to five, we search for the entire unigram in the character-based

dictionary. And because the dictionary contains only 3-grams, 4-grams and 5-grams, combinations that are shorter than three will be automatically identified as 'code'. If the unigram is longer than five we first search for the first five characters, and if those are not found we search for the last five. If no match is found the tag 'code' is given to the unigram. The motivation behind checking the first and last five characters is to try and check certain parts of the word with the character-based language models. The first five characters could be checked as the stem of a word, and the last five as common inflectional suffixes.

As a result, every *n*-gram in the line receives a tag and we then give this line to another function to perform cleartext detection. This is done by checking if the *n*-grams which have the 'text' tag are preceded or followed by the 'code' tag or another 'text' tag: if the *n*-gram is preceded by a *n*-gram with the tag 'code', the current *n*-gram is the beginning of the cleartext and we attach the opening cleartext tag to that word ('<CLEARTEXT'). If it is followed by a *n*-gram with the tag 'code' or if we reach the end of the line, the current *n*-gram is the end of the cleartext and we attach the closing cleartext tag to that word ('>'). As the next step, we identify the language of the cleartext sequences.

3.4. Language Identification

In order to perform language identification, we choose the best performing model for cleartext detection and change the tag assignment function slightly: instead of just giving a generic 'text' tag, the function would look for the word in the dictionaries and if it is found it will retrieve the language with the highest relative frequency and assign it to the *n*-gram. Next, the tagged line is given to another function to decide the language for the whole line, by counting the occurrences for each language in the line. Because bigrams are more relevant than unigrams, we multiply each bigram score by 1 and each unigram score by 0.5. Finally, we output a ranked list of languages with the one with the highest frequency being the first.

4. Results and Discussion

Before we present and discuss the results, we describe the evaluation to measure model performance.

4.1. Evaluation

In order to evaluate our models, we decided to use different measurements. The first measure is to calculate the total line match, where we check for each text output by our models how many of its lines are totally matched with the respective gold standard text. This measure gives us an idea about how well the model is performing overall, without considering specifically the language identification and cleartext detection part. It also gives us an idea of how well the model performs automatic annotation in general.

The second measure is for the calculation of partial line match, where we check if some parts of the cleartext were detected in the line. This measure gives us an idea about how well the model is performing cleartext detection, although partially. Partial performance can be relevant for annotation tasks to detect cleartext quicker.

The third measure is to calculate the standard measures of accuracy, precision, recall and F1-score. These measures give us an idea of how well the model performs cleartext detection and if the models overgeneralize or undergeneralize the detection.

The fourth measure calculates the accuracy in the language identification task, by comparing the tags in each text output by our model with the tags in the gold standard. This measure gives us an idea of how well our models perform in the language identification task.

In order to evaluate language identification accuracy, we iterate through the gold standard file line by line and retrieve the same line in the output file. We first count all the language tags in the gold standard file and then we retrieve all the lines where a language was identified in both the gold standard and our model output files: if the tag in the gold standard text is the same as the one in the output text, we add 1 to the count of the matched tags. We then divide the matched tags by the total number of language tags and multiply by 100 to get the percentage.

4.2. Results of Cleartext Detection

The results from the six models measured on the test set along with the baseline is presented in Table 4. All models with the exception of model 1 outperformed the baseline. Model 4 is the next worst, generating a low partial match, and low precision, but compensate for high a recall. The best performing model is model 6 with F1-Score of 92.46% and the next best is model 5 with F1 score of 91.47%. The results confirm our initial hypothesis that combining character-based bigrams and unigrams can help improving the performance of cleartext detection. It also confirms our hypothesis that having a certain threshold is necessary to avoid overgeneralization, since recall and precision gets better with the introduction of these thresholds.

4.3. The Impact of Symbols

Since ciphers might consists of symbols that co-occur with the plaintext alphabet, we measure model performance on ciphers with various symbol sets. Figure 5 illustrated the model performance measures as F1 on documents with various symbol sets: digits (D), graphic signs (G), letters (L), and various combinations that occur in the test set.

The easiest documents with cleartext to identify turn out to be the ciphers that use graphic signs only. This is not surprising since the cleartexts in the Latin alphabet are clearly distinguishable from the ciphertext with

Model	Total line	Partial line	Accuracy	Precision	Recall	F1
baseline	44.34	80.83	72.12	74.29	84.11	78.90
model 1: W1_TFR	42.72	80.76	71.52	74.32	82.90	78.38
model 2: W1_TL	56.05	93.76	89.75	92.92	82.63	87.47
model 3: W1_TL+W2	51.74	88.99	88.42	90.20	83.33	86.63
model 4: W1_TL+W2+CH345	41.75	67.88	77.36	70.74	93.00	80.36
model 5: W1_TL+W2+CH345_TL	63.16	89.09	93.64	90.48	92.49	91.47
model 6: W1_TL+W2_TL+CH345_TL	67.98	93.85	94.77	92.64	92.28	**92.46**

Table 4: Results (%) for each model on the test set.

graphic symbols and therefore easy to model, as indicated by the high F1 scores of over 90% for almost all models.

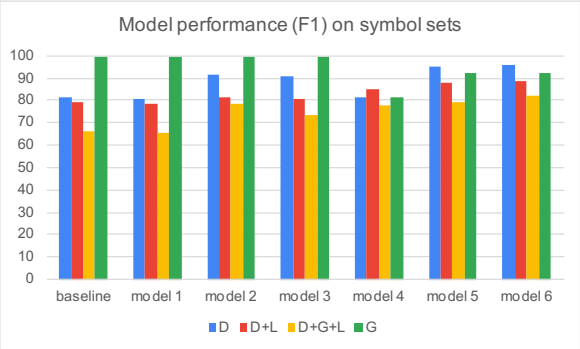

Figure 5: Model performance (F1) on symbol sets: D-digits, G-graphic signs, L-Letters.

The best performing model in general is model 6, scoring 96.03% for ciphertexts using digits, 88.87% for ciphertexts using a combination of digits and letters and 82.17% for ciphertexts using a combination of digits, letters and graphic signs. This model was able to detect the cleartext in the document which presented Unknown cleartext and able to see that no cleartext was present in the document which had none. It is clear that the more combination of symbol sets are used for encryption, the more difficult the identification of cleartext and ciphertext sequences become.

In general, model 6 achieves highest performance for all symbol sets with the exception of graphic signs only. Ciphertexts using a symbol set made of digits were the ones which performed best with model 6. This could be due to the fact that it is easier to detect text in a ciphertext where only numerals are used for code and only letters are used for text. Ciphertexts using a symbol set made of digits and letters and ciphertexts using a symbol set made of digits, letters and graphic signs benefit from model 6 as well, but because it is more difficult to decide if letters are text or code the model performs slightly worse than expected, but reaching a high score nevertheless.

4.4. Results of Language Identification

As the last part of the evaluation, we measure the accuracy of language identification on the line level to account for code-switching in cleartexts with different languages in different lines. The best performing model — model 6 — achieves 44.68% accuracy for the identification of the correct cleartext language.

The models perform differently depending on the languages of the cleartext, as shown in Table 5. The language with the highest accuracy obtained is Dutch with a score of 88.89% followed by Portuguese with a score of 85.29%, although these appear in one document only as part of the test set. If we look at the languages with most data, we can see that Spanish is the best performing language reaching a score of 64.69%.

Because these scores were lower than expected albeit less surprising given that it is hard to automatically guess a language based on such a small context as a few words even for humans, we decided to run the same task on the document level taking into account all cleartext sequences in the document. The reason why we chose this approach is the fact that most ciphertexts in our dataset contain one cleartext language only. Therefore, we chose the most frequent language tag for all cleartext segments in that document and assigned it to the given file. The accuracy for language identification on a document level for the best performing model (model 6) was 70.40%.

Given the results we can conclude that language identification on a document level seems to reach better scores than on a line level. The language with the highest score in language identification accuracy on the document level is Dutch and Portuguese with a score of 100.0%, see Table 5. If we look at the languages with most data, French is the best performing language during testing reaching a score of 100.0%.

When it comes to languages we need to keep in mind certain factors: although Dutch and Portuguese have the best performing results, it is worth mentioning that we had available only one ciphertext for each language. The same goes for Latin and a combination of Latin and French where we had 8 and 6 ciphertexts available, respectively. If we consider the languages that had a bigger amount of data, Spanish is the best performing language. This could be due to the fact that the language model was fairly big, counting around 2.4 million *n*-grams, and there were fewer annotation doubts in the transcriptions, making it easier to detect words. It can be argued that Latin has a bigger language model, counting 20.3 million *n*-grams, and therefore more *n*-

Language	Number of texts	Accuracy-Line	Accuracy-Document
Hungarian	22	16.24	18.18
French	20	57.32	100.0
Italian	20	49.38	75.0
Spanish	17	64.69	94.12
Latin	3	18.45	33.33
Latin/French	2	55.38	50.0
Dutch	1	88.89	100.0
Portuguese	1	85.29	100.0

Table 5: Results (%) for LI for each language on the line and document levels on the test set.

grams useful for the detection of cleartext. Although this is true, we need to remember that Latin texts presented more annotation doubts compared to other languages, making it more difficult to detect words. At the same time, we can find Latin words in other language models as well, since Latin was widely used also in texts mainly written in another language. Hungarian presented similar characteristics as Latin, with more annotation doubts in its transcriptions than other languages, but at the same time it also had a smaller language model, counting 1.6 million *n*-grams. French and Italian follow Spanish and this can be due to the size of data and low amount of annotation doubts.

5. Conclusion

In this paper, we addressed the problem of detecting cleartext in a ciphertext and identifying its language. In order to perform this task we used the language models available on the HistCorp platform, and created two dictionaries: one containing unigrams and bigrams on a word level and one containing 3-grams, 4-grams and 5-grams on a character level. We then built our baseline model using unigrams only and compared it against 6 models which used unigrams only, or a combination of unigrams and bigrams, or a combination of unigrams and bigrams on word-level and 3-grams, 4-grams and 5-grams on character-level. We experimented with different thresholds for all order *n*-grams both on a word level and on a character level. Our intuition was that by combining unigrams, bigrams and characters while having a threshold on each of them, the model would perform better. Our idea was that the threshold could filter out items which could have been misunderstood as cleartext when they were code, or vice versa. In order to perform the cleartext detection task we checked the text line by line. If the *n*-grams analyzed were present in the dictionaries and depending on the model, we gave a 'text' tag or a 'code' tag for each text sequence. In order to evaluate the models, we used different measurements such as total line match, partial line match, accuracy, precision, recall and F1 to have a complete overview and understanding of how the models were performing.

Our results confirmed our hypothesis with the model using a combination of unigrams, bigrams on a word level and 3-grams, 4-grams and 5-grams on a character level, reaching the highest F1-score of 92.06%. A threshold was used on all *n*-grams: unigrams on a word level and all *n*-grams on a character level had a threshold on items that presented at least one digit, whereas bigrams on a word level had a threshold on items that presented only digits.

For the language identification task, the results for each language were quite diverse, probably due to the differences in the size of the language models. We noticed that on a line level Spanish reached good results among the languages which had a more balanced ratio of training and test set (64.69%).

Future research should consider using a combination of the best performing model in this paper with 3-grams on a word level and see if a threshold could further improve the performance of the model. We believe that including higher order *n*-grams can help the model to detect more difficult combinations of words such as *27th August 1679*, where with a lower order *n*-gram *27* and *1679* could be detected as code.

It would be of interest that future research investigates how to deal with doubts in the transcriptions in a deeper way. A suggestion could be to take the words that the annotators were unsure about and try to find the most similar one in the language models. This approach could improve both cleartext detection and language identification since it will reduce the chances of these words being tagged as code.

Future research might also apply machine learning algorithms to this task, but only in the event that more data would be available. Regarding the language identification task, a research suggestion could be to create equally sized language models for all languages, so that words have a lower chance to be assigned to the wrong language because of lower relative frequencies due to lack of data.

All in all, we find the results promising, especially the cleartext identification task while language identification of a couple of words remains challenging.

6. Bibliographical References

Aldarrab, N., Knight, K., and Megyesi, B. (2017). The borg cipher. `https://cl.lingfil.uu.se/~bea/borg`. Accessed: 2020-01-31.

ASV. (2016a). Reproduced excerpt from the Vatican Secret Archive, i. 1025, Segretario di Stato, Francia, doss. 117, DECODE link: https://decrypt.org/decode Record ID 198.

ASV. (2016b). Reproduced excerpt from the Vatican Secret Archive, i. 1025, Segretario di Stato, Francia, doss. 64-8, DECODE link: https://decrypt.org/decode Record ID 69.

Beesley, K. R. (1988). Language identifier: A computer program for automatic natural-language identification of on-line text. In *Proceedings of the 29th annual conference of the American Translators Association*, volume 47, page 54.

Chanda, A., Das, D., and Mazumdar, C. (2016). Columbia-jadavpur submission for emnlp 2016 code-switching workshop shared task: System description. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 112–115.

(2011–2020). The Copiale Cipher. `https://cl.lingfil.uu.se/~bea/copiale`. Accessed: 2020-01-31.

Mona Diab, et al., editors. (2014). *Proceedings of the First Workshop on Computational Approaches to Code Switching*, Doha, Qatar. Association for Computational Linguistics.

Mona Diab, et al., editors. (2016). *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, Austin, Texas, November. Association for Computational Linguistics.

Federico, M., Bertoldi, N., and Cettolo, M. (2008). Irstlm: an open source toolkit for handling large scale language models. In *Ninth Annual Conference of the International Speech Communication Association*.

Megyesi, B., Blomqvist, N., and Pettersson, E. (2019). The decode database: Collection of historical ciphers and keys. In *The 2nd International Conference on Historical Cryptology, HistoCrypt 2019, June 23-26 2019, Mons, Belgium*, pages 69–78.

Megyesi, B., Esslinger, B., Fornés, A., Kopal, N., Láng, B., Lasry, G., Leeuw, K. d., Pettersson, E., Wacker, A., and Waldispühl, M. (2020). Decryption of historical manuscripts: the decrypt project. *Cryptologia*, pages 1–15.

Megyesi, B. (2020). Transcription of historical ciphers and keys. In *the 3rd International Conference on Historical Cryptology*. Linköping University Electronic Press.

Miller, B. N. and Ranum, D. L. (2006). *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle and Associates.

Pettersson, E. and Megyesi, B. (2018). The HistCorp Collection of Historical Corpora and Resources. In *Proceedings of the Digital Humanities in the Nordic Countries 3rd Conference*, Helsinki, Finland, March.

Pettersson, E. and Megyesi, B. (2019). Matching keys and encrypted manuscripts. In *The 22nd Nordic Conference on Computational Linguistics (NoDaLiDa'19)*. Linköping University Electronic Press.

Samih, Y., Maharjan, S., Attia, M., Kallmeyer, L., and Solorio, T. (2016). Multilingual code-switching identification via lstm recurrent neural networks. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 50–59.

Shirvani, R., Piergallini, M., Gautam, G. S., and Chouikha, M. (2016). The howard university system submission for the shared task in language identification in spanish-english codeswitching. In *Proceedings of the second workshop on computational approaches to code switching*, pages 116–120.