A Massive Machine-Learning Approach For Classical Cipher Type Detection Using Feature Engineering

Ernst Leierzopf[1]*, Nils Kopal[2], Bernhard Esslinger[2],
Harald Lampesberger[1], Eckehard Hermann[1]
[1]University of Applied Sciences Upper Austria, Hagenberg, Austria
[2]University of Siegen, Germany
*e.leierzopf@gmail.com

Abstract

Cryptanalysis of enciphered documents typically starts with identifying the cipher type. A large number of encrypted historical documents exists, whose decryption can potentially increase the knowledge of historical events. This paper investigates whether machine learning can support the cipher type classification task when only ciphertexts are given. A selection of engineered features for historical ciphertexts and various machine-learning classifiers have been applied for 56 different cipher types specified by the American Cryptogram Association. Different neuronal network models were empirically evaluated. Our best-performing model achieved an accuracy of 80.24% which improves the current state of the art by 37%. Accuracy is calculated by dividing the total number of samples by the number of true positive predictions. The software-suite is published under the name "Neural Cipher Identifier (NCID)".

1 Introduction

Historical records show that encryption is about as old as scripture itself. The earliest documented use of cryptography can be traced back to the Old Egyptian Empire in the third millennium BC (Lieven, 2007). In ancient history, cryptography was mainly used by the aristocracy and the military. In principle, classical ciphers can be divided into substitution ciphers and transposition ciphers. With simple substitution, each letter is substituted with a different one from the alphabet. Homophonic substitution replaces letters by several different substitutes, so the ciphertext alphabet is bigger than the plaintext alphabet. Transposition ciphers mix (permute) the letters of the plaintext into a quasi-random order. There are also ciphers combining substitution and transposition like ADFG(V)X (Friedman, 1941).

A typical cryptanalysis method for classical substitution ciphers is frequency analysis. Here, the frequencies of single or groups of multiple ciphertext symbols are counted and then compared to the frequencies of the assumed plaintext language. Then, based on the different frequencies in the plaintext language, assumptions of which letter was replaced by which symbol, can be made. Knowledge of the used cipher type allows the application of cipher-specific and heuristic algorithms to find the plaintexts more precisely. For example the Kasiski examination (1863) of the Vigenère cipher takes advantage of the fact that, by chance, repeated words are sometimes encrypted using the same key letters and therefore give indication for the possible key lengths.

The goal of this research is to determine how cipher type detection can be improved with machine learning approaches like feedforward neural networks (FFNN), decision trees (DT), random forests (RF) and naïve Bayes networks (NBN) using newly calculated statistics in a massive feature engineering approach (see Section 3.7). A systematic, exhaustive evaluation over all American Cryptogram Association (ACA) ciphers (2005) has been performed. To achieve the best results, multiple optimizers, activation functions and features were implemented and evaluated with several relatively unparameterized neural networks. The identified features are mainly based on previous work starting with Kopal's prototype (2020) for the MysteryTwister[1] challenge "Cipher ID", and on the implementations from Bion (Mason, 2021).

[1]MysteryTwister: https://www.mysterytwisterc3.org/

The result of this work is the software suite "Neural Cipher Identifier (NCID)" (Leierzopf, 2021), which also will be available online.[2] It can be used for training and evaluation of neural networks with different classifiers like FFNN, DT and NBN. A potential use case for the software suite could be the DECRYPT project, with the aim to offer a working infrastructure for researchers enabling the collection, automated digitization, analysis, and decryption of encrypted historical documents (Megyesi et al., 2020).

This paper is structured as follows. The next chapter discusses all important related work for cipher type detection with neural networks. The third chapter describes all implemented cipher types, data generation procedures, feature selection and different classifier architectures. The results of the empirical evaluation are summarized in Chapter 4. Chapter 5 concludes this paper.

2 Related Work

The search for related work included classification of classical ciphers as well as modern ciphers. The idea was that recognition methods, which work on modern ciphers, most certainly also work on classical ciphers. It is state-of-the-art to use the same datasets for training to achieve better comparability to other work. Unfortunately no such standard dataset exists for the field of cipher type detection.

Nuhn and Knight achieved remarkable results in the area of the classification of classical ciphers with neural networks and were the benchmark (2014). The researchers trained a neural network with a linear classifier and a Stochastic Gradient Descendent (SGD) optimizer with default parameters for 50 ACA ciphers. An accuracy of 58.5% was achieved by using a quadratic loss function and adaptive learning rates with 1 million ciphertexts and 20 epochs. According to Nuhn and Knight, squared features have not improved accuracy. They implemented 55 features from Bion and developed three features themselves. The random text lengths without defined ranges of lengths are a major drawback in the comparability of their work. The reason for this assumption is that most features rely on statistical calculations, which are more precise for longer texts.

Results from the work of Sivagurunathan et. al (2010), where the three classical ciphers Playfair, Hill and Vigenère were analyzed with a simple neural network, coincide with the results of Kopal (2020). Both discovered the difficulty of classifying (distinguishing) the Hill and Vigenère ciphers, because of their similar statistical values.

A multi-layer classifier has been introduced by Abd and Al-Janabi (2019) to classify plaintexts and ten different cipher types. The impressive results of over 99% accuracy are lessened by the enormous ciphertext length of about one million characters, which is the equivalent of an average book with 500 pages. Ciphertexts with these lengths are seldom. The greatest part of original historic encrypted manuscripts are only between a few lines and some pages of ciphertext long.

Krishna (2019) developed approaches that have not yet been used by other authors for the four ciphers Simple Substitution, Vigenère, Transposition and Playfair. An important point for comparison is that the Hill cipher was not used here. The first approach, a support vector machine (SVM), uses the ciphertexts of length 10 to 10,000, which are mapped in a number range, as training data. The SVM uses the implementation of the Sklearn Library[3] with a 10-Fold StratifiedKFold Cross-Validation and a One-vs-Rest Classifier for the calculation of the confusion matrix. This means that 9 out of 10 datasets were used for training and 1 dataset for testing in order to find the most suitable class. In the second and third approach, a Hidden Markov Model (HMM) was trained for 1000 ciphertexts per class and used by means of conversion as input for a convolutional neural network (CNN) in the second approach and an SVM in the third approach. The first approach achieves an accuracy of 100% with a text length of 200, the second 71% with a text length of 155 and the third 100% with a text length of 155.

Zhao et al. (2018) extracted 54 features from 15 different NIST 800-22 (Rukhin et al., 2010) randomness tests. The efficiency was tested in several 10-fold-cross-validation SVM one-to-one classifiers for six modern block ciphers (AES, Blowfish, Camellia, DES, 3DES and IDEA). The result was that 42 features gave better results than random guessing, i.e. 50%. 12 of these features even provide a recognition rate of over 60%.

Tan and Ji (2016) developed a very similar model with the five modern ciphers AES, Blowfish, DES, 3DES and RC5. The experiments are

[2] https://www.cryptool.org/ncid

[3] Sklearn Library: https://scikit-learn.org/stable/

Author	# Features	Accuracy in %	Text Length	Dataset Size	Epochs	# Cipher Types	Cipher Category	Technology
Nuhn	58	58.50	random	1,000,000	20	50	classical	Vowpal Wabbit
Nils Kopal	4	90	100	4,500	20	5	classical	FFNN
Sivagur.	12	84.75	1,000	900	N/A	3	classical	FFNN
Abd	7	99.60	1,000,000	N/A	500	11	classical	3-Level-Classifier
Krishna	N/A	100	155	4,000	N/A	4	classical	SVM, Hid. Markov M.
Zhao	54	47.8-89.5	512,000	6,000	N/A	6	modern	One-vs-One SVM
Tan	N/A	39	100,000	1,100	N/A	5	modern	SVM
Manjula	10	72.20	1-2,000	1500	N/A	11	modern	DT
Chandra	46	80	12,800	1,000	N/A	3	modern	One-vs-One FFNN

Table 1: Summarized results and attributes of related work

carried out in this work with two scenarios: Once the same key material for training and test data and the other time with different key material. For the same key, the result is 85% from 20 kB of data and 96% from 100 kB of data. For different key lengths with the same amount of data it is 35% or 39%. The parameters used by the SVM were not explained in more detail.

Manjula and Anitha (2011) designed a C4.5 classifier for eleven modern ciphers and achieved a recognition rate of over 70% for ciphertexts with a variable length of 1-2000 bytes. The C4.5 algorithm creates a decision tree based on the information gain ratio. A total of ten features were designed, seven of which are based on the maximum entropy of different characters. Further features are the entropy of all characters, the correlation coefficient of capital letters and the length of the ciphertext, since the expected entropies depend on this.

Different algorithms for the one-to-one classification, i.e. a comparison of individual modern stream and block ciphers, were presented by Chandra et al. (2007). The tested neural network architectures were back propagation, back propagation with momentum, resilient propagation, scaled conjugent gradient, conjugent gradient with Powell-Beale restarts and conjugent gradient with Polak-Ribiere update. On average, all algorithms achieved an accuracy of over 80%, but resilient propagation was able to achieve over 6% better results, especially comparing one stream cipher with another stream cipher. The training was carried out with texts with a length of 12.8 kB and 46 features that are not described in detail.

Table 1 summarizes the state of the art with respect to the number of features, the self-reported accuracy, the utilized text lengths for evaluation and the training dataset size in the respective paper.

3 Neural Cipher Identifier

In this paper general classifier architectures are referred to as classifiers. Trained instances of these classifiers are called models. The selected architectures and algorithms of the models were, more or less, biased by the knowledge of the authors, and the hyperparameters were set to default values without optimization.

A common theme in the related work is that features are selected from one or a few sources and not further questioned or tested. These features are often incomplete and correlated to other features, which can even make the models worse. By selecting features based on individually tested results and implementing and optimizing multiple feature engineering machine learning approaches, better results for more cipher types can be expected. This approach has been implemented in this paper. Every test used newly generated data to prevent specialization on a specific dataset.

3.1 Implemented Cipher Types

This work is based on Kopal (2020), who analyzed the five ciphers, simple monoalphabetic substitution, columnar transposition, Vigenère, Hill and Playfair with an FFNN. As a result from previous work, an FFNN with five hidden layers and a hidden layer size of

$$2 \cdot \frac{input\_layer\_size}{3} + output\_layer\_size$$

is used as the starting point of research (further on called **"baseline reference model"**). In the first step, the solution was expanded by adding interfaces, cipher implementations, a custom data loader and a testsuite for all classes. Training and test data is generated on-the-fly.

For the test setup, all Bion features plus the already existing features from previous work together with a selection of 55 of the 60 ACA ciphers and plaintext were used. The ciphers Twin Bifid and Twin Trifid were excluded, because they combine two ciphertexts, Incomplete Columnar Transposition and Interrupted Key were also excluded, because they are indistinguishable ciphers. Syllabary (Friedman, 2012) was not implemented, because it was invented 2012 and does not fit into the classical cipher period. Table 2 shows the ciphers used during this evaluation.

amsco	grandpre	per. gromark	ragbaby
autokey	grille	phillips	railfence
baconian	gromark	phillips rc	redefence
bazeries	gronsfeld	plaintext	route transp.
beaufort	headlines	playfair	running key
bifid	homophonic	pollux	seriatedpfair
cadenus	key phrase	porta	slidefair
checkerboard	mnmedinome	portax	swagman
col. transp.	morbit	progkey	tridigital
condi	myskowski	quagmire1	trifid
cmbifid	nicodemus	quagmire2	trisquare
digrafid	nihilist subst.	quagmire3	two square
foursquare	nihilist transp.	quagmire4	variant
fract. morse	null	numbered key	vigenère

Table 2: All 56 implemented ACA ciphers

To get comparable results with different architectures, the training and validation text length is fixed to 100 characters, after all non-alphabet characters are filtered. According to the American Cryptogram Association (2005), all ACA ciphers need 40 to 220 characters to be broken.

3.2 Keywords

For historical reasons, all ACA ciphers, whose keys do not consist of digits, do not choose the key words and alphabets at random, but rather use English words. So called key alphabets use one keyword and fill the rest of the alphabet in the alphabetical order. This allows the following three training scenarios to be defined and sorted by their classification difficulty:

1. Keywords are chosen from a dictionary. Key alphabets use key words from a dictionary and the rest of the alphabet is arranged in the correct order.
2. All characters of keywords are chosen at random. Key alphabets use keywords with all characters being chosen randomly and fill the rest of the alphabet in the correct order.
3. All characters of keywords are chosen at random. Key alphabets are arranged randomly.

By default, all tests were run in the second scenario which use keywords with all characters chosen randomly and key alphabets are arranged in the correct order after the keyword. Filling the rest of the alphabet in the correct order is a major weakness of each cipher. However, historically ciphers were used in the first scenario with a word chosen from a dictionary. At the time of invention this procedure offered enough security. Compared to scenario 1, the advantage of the second scenario is that the model is less likely to be overfitted due to the lack of different keywords for specific lengths and it should be more general and more secure than with predefined keywords.

3.3 Optimizer Selection

Before testing new architectures some tests were made beforehand. The algorithm used to determine the weights after every training cycle is referred to as optimizer. The optimizer was selected by the best result of empirical test runs with the default parameters. SGD with Momentum = 0.9, RMSprop, Adam, Adadelta, Adagrad, Adamax and Nadam were tested. To find out which of the seven optimizers is the best for our scenario, each model was trained with a different optimizer with 100 million data records, i.e. 1.8 million per cipher. Ciphers which need keywords were trained with the key lengths 5 to 8 as these lengths were typical at the time of invention. The rest was trained with no keywords and the same amount of data. A baseline reference model with plaintexts of the exact size of 100 characters and the second training scenario were used for the comparison.

Optimizer	Accuracy in %	Top 3 Accuracy in %	Training Time	Converge?
SGD with Momentum	64.78	81.50	4h 21m	Yes
RMSprop	69.96	84.60	4h 27m	Yes
Adam	72.97	87.18	4h 21m	No
Adadelta	48.04	68.82	4h 23m	No
Adagrad	56.31	74.74	4h 23m	No
Adamax	73.71	87.80	4h 25m	No
Nadam	72.42	86.91	4h 33m	Yes

Table 3: Results of the comparison of 7 optimizers

From the results in Table 3 it can be seen that Adam and Adamax deliver the best results in terms

of accuracy with default parameters. The training time of the model is very close for all optimizers which can be attributed to the preprocessing time on the CPUs being greater than the training time on the GPUs. For better comparability, further tests and the search for the best hyperparameters are carried out with the Adam algorithm.

3.4 Activation Functions

Activation functions are mathematical functions which are used to adapt the weights in a neural network. In order to be able to determine how the training corresponds to different activation functions, the baseline reference model was trained up to convergence with 10 different activation functions. The exponential function is an exception in which only one hidden layer was used, as otherwise the loss cannot be calculated. These results were calculated after the selection of the best features, which is described later on. Using the activation functions in Keras (2021), the results of the activation function comparison can be seen in Table 4.

Function	Accuracy in %	Top 3 Accuracy in %	Training Time	Converges after Mio. Iterations
ReLU	74.70	88.94	16h 50m	146
Leaky ReLU	72.64	87.45	12h 30m	99
Parametric ReLU	75.18	89.02	19h 29m	152
Sigmoid	72.32	87.01	1d 22h	385
tanh	65.48	81.87	9h 33m	68
ELU	68.51	84.18	10h 24m	76
SELU	67.54	83.58	13h 48m	103
Exponential	70.10	85.62	17h 54m	140
Swish	70.32	86.07	20h 33m	150
RBF	1.59	4.92	3h 30m	16

Table 4: Results of the activation function comparison

With the exception of the Parametric ReLU function, the ReLU function delivered the best results in terms of accuracy and training time. Due to the more reliable results of the ReLU function it was preferred over the more complex Parametric ReLU function in further tests. The exponential function delivered impressive results with only one hidden layer.

3.5 Data Generation

Figure 1 shows the training process of the cipher classification model. 14 GB of English texts of the Gutenberg Project[4], which is free to use for research purposes, were used as dataset for training and validating of the models. Loading and preprocessing of the features is done by an own data loader and is described in more detail in the next paragraph. After the training process is completed, the model is saved and evaluated.

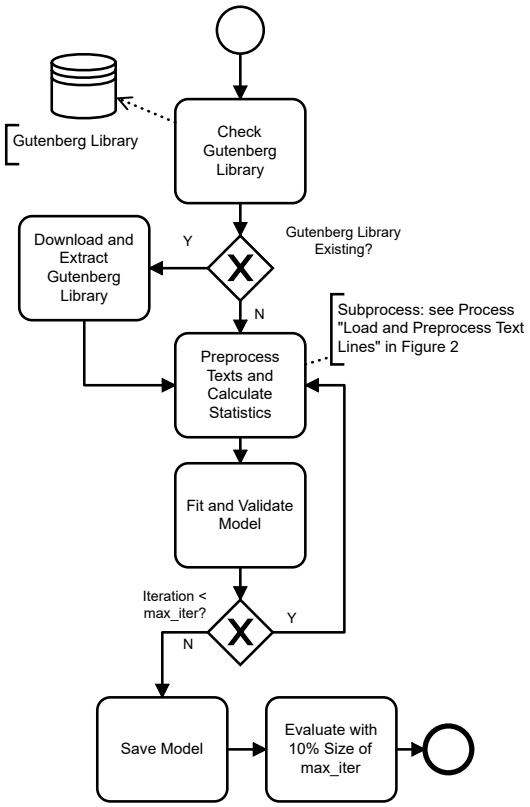

Figure 1: Training process

Figure 2 shows the data loader process described in the last section. This process loads one or multiple lines of text, depending on the defined ranges, from the given dataset, and adapts them with the appropriate filter function for the specific cipher. As the final length after a text is read from file can not be determined due to the filtering of non-alphabetic characters, lines are read in loops. For example, the filter of the Playfair cipher replaces all J characters with I characters. The length of the entire text can be set using command line arguments. It can be assumed that longer messages are easier to classify because the calculated features are more meaningful. The process described must be carried out until the number of plaintexts generated equals the size of the required dataset, which is defined as a parameter in the program itself, divided by the sum of the ciphers and

[4]https://www.gutenberg.org/

their configured key lengths. This means that each plaintext can be used once for each cipher with each of the configured key lengths for training the model. After enough lines of text are available, several processes, so-called workers, are started in figure 2 to calculate the features in parallel.

3.6 Features

The selected features can be divided into the following groups:

- frequency statistics (e.g. unigrams, bigrams)
- distribution statistics (e.g. IoC)
- binary features (e.g. HAS_J, HAS_X)
- cipher-specific features (e.g. A_LDI)

Abbr	Term	Description
SDD	Average Single Letter – Digraph Discrepancy Score	This feature uses a table of the differences between unigrams and bigrams. The score is calculated by adding each value at the position of the first letter in the alphabet times 26 plus the position of the second letter in the alphabet from the SDD table. The score is then divided by the length of the text minus 1. For normalization the scores are divided by 10.
CHI^2	Chi Square	With the Chi^2 function, the deviation from the distribution of English letters, which is known, can be calculated. This value is divided by 100 to be normalized.
DIC	Digraphic Index of Coincidence	Sum of all probabilities of the occurrence of two identical pairs of characters in a text times 1000.
DBL	Double Letter	Binary value about the occurrence of a double character in an even place and that the total length is even.
AUTO	Estimated Auto Correlation	Autocorrelation is useful in identifying repeating patterns in a sequence. Due to the different lengths of the ciphertexts (the Null cipher has ciphertexts a maximum of 10 times as long as plain texts), the remaining data points must be filled with 0.
FREQ	Frequencies	Recursive calculation of the probability of occurrence up to and including bigrams.
HAS_0	Has Digit 0	Binary value based on the occurrence of the digit 0.
HAS_H	Has Hash	Binary value based on the occurrence of the # sign.
HAS_J	Has Letter J	Binary value for the occurrence of the letter J.
HAS_X	Has Letter X	Binary value for the occurrence of the letter X.
HAS_SP	Has Space	Binary value based on the occurrence of the space character.
IoC	Index of Coincidence	Sum of all probabilities of the occurrence of two identical characters in a text.
LDI	Log Digraph Score	Bigrams in a text are searched for in a list of precalculated English letter frequencies and added up. The average of this sum is the score. At Bion, the real numbers are used instead, but these are too large values, which is why the probability of occurrence divided by 10 is more suitable.
A_LDI, B_LDI, P_LDI, S_LDI, V_LDI, PTX	Log Digraph Score for Autokey, Beaufort, Porta, Slidefair, Vigenère, and Portax	The LDI calculates this set of Vigenère statistics for different ciphers by converting the ciphertexts with the respective shift functions. The score is divided by 1000. For ciphertexts that contain characters other than letters, the PTX feature is 0.
LR	Long Repeat	Percentage of characters that are repeated exactly three times. For this purpose, all the same characters are counted for each character from position +1. The root of this result is divided by the length of the text.
BDI	Max Bifid DIC for Periods 3-15	As in the Bifid cipher, texts are read in periods of 3-15 and the DIC is calculated from this. The highest score is divided by 1000 and returned. For ciphertexts that contain characters other than letters, this feature is 0.
CDD	Max Columnar SDD Score for Periods 4-15	As in the columnar transposition cipher, texts are read in periods and the SDD score is calculated for them. The result of this feature is the maximum SDD score divided by 1000. This feature is 0 for ciphertexts that contain characters other than letters.
MKA	Max Kappa	Texts are shifted by p to the right for Periods 1-15. The remaining p characters are padded with values that are not contained in the text (e.g. -1). The result of this statistic is the maximum percentage of match between the moved text and the original text.
NIC	Max Nicodemus IC	Texts are divided into periods 3-15. The highest NIC is calculated by dividing and reading the text as with the Nicodemus cipher. The highest value is returned.
SSTD	Max STD Score for Swagman Periods 4-8	As in the Swagman cipher, texts are read in periods and the STD score is calculated. The result of this feature is the maximum STD score divided by 100.
MIC	Maximum Index of Coincidence	Texts are divided into periods 1-15. The highest IoC of all subgroups is calculated by dividing the text into p groups. Each group consists of all characters spaced p. If p = 3 there are 3 groups, whereby the first group contains every third character starting with 0; the second group every third character starting with 1 and the third group every third character starting with 2. The highest value is returned.
NOMOR	Normal Order	The frequency of each character is calculated and sorted by size. The normal order is the sum of the distances of all characters from their normal position divided by 1000.
PHIC	Phillips IC	Calculates the IC using a fixed column size = 5 and a fixed period = 8. The result is multiplied by 10. For ciphertexts that contain characters other than letters, this feature is 0.
REP	Repetition Feature	This feature is adopted from Nuhn and Knight (2014). It consists of the normalized number of exactly n times occurring identical characters for $2 \leq n \leq 5$. The normalization is calculated by dividing through the total number of repetitions.
ROD	Repetition Odd	Percentage of odd-spaced repeating characters to the sum of repeating characters. For this purpose, all the same characters are counted for each character from position +1. The result is sum_odd / sum_all
RDI	Reverse Log Digraph	Bigrams in a text are searched for in a list of precalculated English letter frequencies and added up, but the order of the letters is reversed, e.g. AB -¿ BA. The average of this sum is the score. With Bion, the real numbers are used instead, but these are too large values, which is why the probability of occurrence divided by 10 is more suitable.
SHAN	Shannon's Entropy Equation	Entropy is a measure for determining the information content of a text. Basically, a higher entropy indicates that data is encrypted. This value is divided by 10.

Table 5: Feature definitions

The value ranges of the features are normalized to [0..1] so that small changes in a feature with a higher value range do not have disproportionate

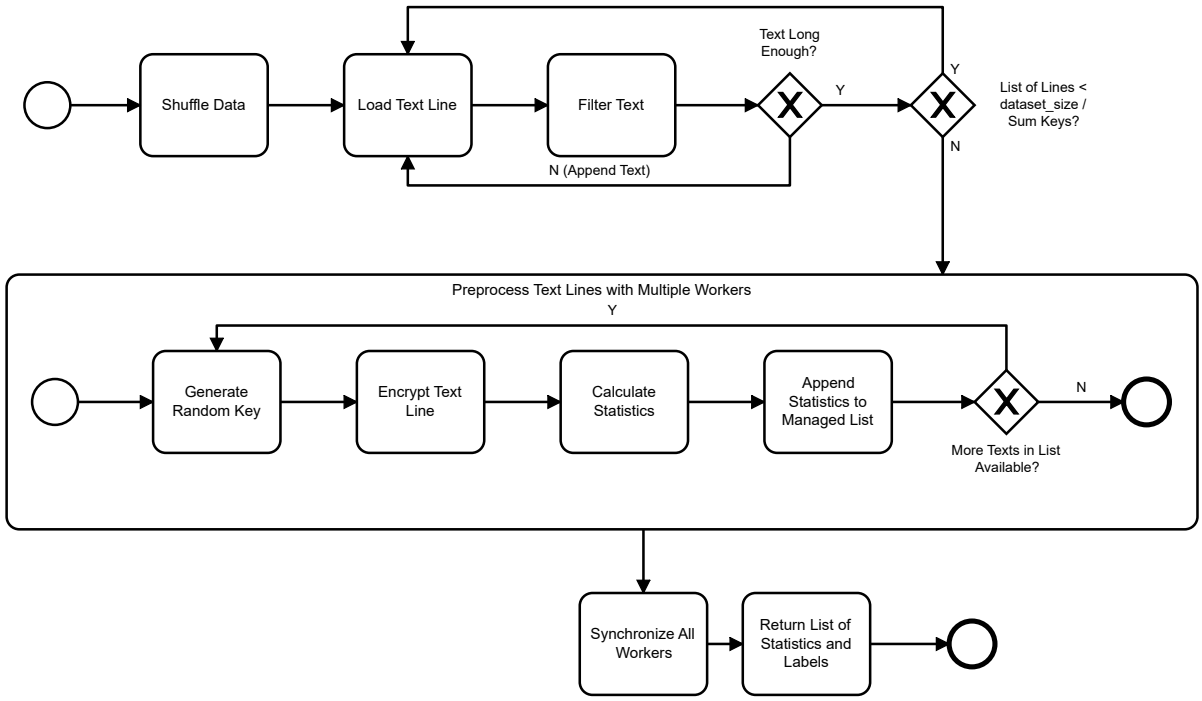

Figure 2: Load and preprocess text lines

effects on the decision and on the evaluation of other features in the learning process. Every feature was calculated 1000 times. The average calculation time contributed to the decision whether a feature was included or not. Totally 28 features were implemented and out of these 20 were selected after extensive testing (see Chapter 4). The tests were run evolutionary. This method does not need as much testing as a grid search. Every feature which achieves better accuracy than the last configuration is included in the following test. Table 5 describes all implemented and tested features. The selection is based on tests with adding one feature at the time and checking the difference in the results.

3.7 Classifier Architectures

Feature engineering classifiers have the property that selected features are provided as input for the model through expert knowledge. With regard to text classification, features can be properties like HAS_J, which checks if a J can be found in the text, and statistics like the index of coincidence. An essential advantage of the feature-engineering method is that known weaknesses of ciphers can be modeled as features and trained in the neural network. The greatest disadvantage of this method is the initial effort required to design and implement the features mentioned.

Feedforward neural networks (FFNN) are based on differentiable activation functions and the finding of the local minimum for the gradient descent error function.[5] The structure of an FFNN consists of one or more layers. The layers between the input and output layers are called hidden layers. A neuron never has connections to other neurons in the same layer. The output of one layer is used as the input of the next layer. The goal of the training phase is to calculate the optimal multiplier (weight) for every connection to minimize the error (loss) by using a small factor (learning rate). The weight is adjusted with a small part of the calculated loss after each update iteration. The complexity of a model is determined by the number and width of the hidden layers and must not be too simple or too complicated. The statistics bias and variance are mostly used to evaluate models. Bias refers to errors due to relationships that have not been learned. The variance is the sensitivity to training data. A model that is too simple can be recognized by a high bias and a low variance (underfitting). A model that is too complicated gets a low bias and a high variance (overfitting) due to the irrelevant features of the data. (Tino et al., 2019)

[5]Gradient descent error function: https://towardsdatascience.com/an-overview-of-the-gradient-descent-algorithm-8645c9e4de1e

Decision tree (DT) algorithms construct binary trees from data for decision making. Each DT has a root node, internal nodes and leaf nodes. Ultimately, the decision always takes place in a leaf node. DT are prone to overfitting and therefore misclassification, which is why the structure of the DT is built up in two phases: training and pruning. During training, the tree is built with all the nodes. The task of the pruning phase is to remove rarely used nodes in order to improve the accuracy and runtime of the DT. (Anyanwu and Shiva, 2009)

DT can be used in serial algorithms (e.g. C4.5 or CART) and in parallel algorithms (e.g. RF). RF consist of multiple incomplete DT with randomly selected features from the entire feature map. These DT are called estimators.

Naïve Bayes networks (NBN) are based on the assumption that all attributes or features of data are completely independent of one another. NBN classifiers make decisions by using the maximum a posteriori estimation with the individual attributes. (Huang and Li, 2011)

Depending on whether the classification problem requires one or more classes, a decision function must be implemented. In the case of clear decision-making problems, in most cases the class with the greatest probability is chosen. Classification problems with multiple outcomes can be classified using a threshold method. Basically, a distinction can be made between Bernoulli and multinomial NBN. Bernoulli NBN can only use binary features. In contrast, multinomial NBN are able to use discrete data for classification.

4 Empirical Evaluation

The best feature map combination from Table 5, which consists of the 20 features SDD, DIC, FREQ, HAS_0, HAS_H, HAS_J, HAS_X, HAS_SP, IoC, LDI, LDI_STATS, LR, BDI, PTX, MKA, NIC, MIC, NOMOR, PHIC and ROD, led to 80.24% accurracy with the FFNN classifier.

Simple decision trees (DT) achieved an accuracy of 61.68%. Random forest classifiers (RF) achieved 71.15% accuracy with 1000 estimators and a maximal depth of 30 without using the LDI_STATS feature. RF achieved good results with a fraction of the training time and data. An essential drawback of RF are the enormous memory requirements, which peaked at about 350 GB, and a very large model to be saved. Therefore, a small RF model with only 100 estimators and setting the parameters minimal samples leaf and split to 10 achieved 74.35% with only 6.4 GB of space, using the LDI_STATS feature. Naïve Bayes networks did not perform well for this specific problem with the provided features. They only achieved 54.17% accuracy. Overall, FFNN achieve the best results for feature engineering classifiers.

Table 6 shows a comparison between all four tested models and Nuhn's work concerning accuracy and memory requirements. All of these models, excluding Nuhn's, used 20 features and a plaintext length of 100 for 56 ciphers. Nuhn's work has been selected to compare with, because it is the most comparable work from Table 1 to this one. The other authors from Table 1 used a much smaller set of different cipher types.

Technology	Accuracy in %	Memory Usage in MB
Nuhn's Vowpal Wabbit	58.50	N/A
FFNN	80.24	45
DT	61.68	300
RF	74.35	6,400
NBN	54.17	2

Table 6: Summarized results compared to Nuhn's work

5 Conclusion

Random English plaintexts were encrypted with 56 different cipher types specified by the American Cryptogram Association. The task was to train models which can be used to determine the cipher type of given ciphertexts. In the feature testing and hyperparameter optimization phases more than 100 models were systematically trained, each one having a computing time of about one day on a Nvidia DGX-1 V100 deep learning machine. As a result, the best configurations for different types of machine learning models were found. In summary, feedforward neural networks (FFNN) provide the best models in terms of accuracy. Random forest classifiers (RF) on the other side only need small amounts of data with about 3 million records to deliver good results in comparison to 200-250 million records with the FFNN.

Further work in this field could include training models with texts from different languages or with texts including errors, as these likely happened in historical documents. Another related question is, whether different features can help in finding

the key of a ciphertext and if feature engineering is the best approach for this problem. More modern ciphers used in World War II can also be implemented and tested with the existing classifiers. This work can be further extended by testing if feature-extracting neural networks can achieve similar or even better results without engineering and testing features. Another extension would be to train and apply these classifiers for modern ciphers.

Acknowledgements

This work has been supported by the Swedish Research Council (grant 2018-06074, DECRYPT – Decryption of historical manuscripts) and the University of Applied Sciences Upper Austria for providing access to the Nvidia DGX-1 deep learning machine.

References

A. Abd and S. Al-Janabi. Classification and Identification of Classical Cipher Type using Artificial Neural Networks. *Journal of Engineering and Applied Sciences*, volume 14, 2019.

American Cryptogram Association. Cryptogram. 2005. https://www.cryptogram.org/.

M. Anyanwu and S. Shiva. Comparative Analysis of Serial Decision Tree Classification Algorithms. *International Journal of Computer Science and Security*, volume 3, June, 2009.

B. Chandra, P. Pallath, P. Saxena, and S. Kant. 3rd Indian International Conference of Artificial Intelligence (IICAI-07). *Neural Networks for Identification of Crypto Systems*, pages 402–411, New Delhi, India, January, 2007.

W. F. Friedman. Military cryptanalysis. volume 61, 1941.

Robert J. Friedman. The syllabory cipher. *Cryptogram*, May-June, 2012.

Y. Huang and L. Li. Naïve Bayes classification algorithm based on small sample set. *IEEE International Conference on Cloud Computing and Intelligence Systems*, pages 34–39, Beijing, China, September, 2011.

Friedrich Kasiski. *Secret writing and the Art of Deciphering*. E. S. Mittler und Sohn, Kingdom of Prussia, 1863.

Keras. Keras. 2021. https://keras.io/api/layers/activations/.

Nils Kopal. Of ciphers and neurons–detecting the type of ciphers using artificial neural networks. *Proceedings of the 3rd International Conference on Historical Cryptology HistoCrypt 2020*, number 171, pages 77–86. Linköping University Electronic Press, 2020.

N. Krishna. Classifying Classic Ciphers using Machine Learning. Master's thesis, San Jose State University, California, USA, May, 2019.

E. Leierzopf. NCID - Neural Cipher Identifier. 2021. https://github.com/dITySoftware/ncid.

A. Lieven. *Grundriss des Laufes der Sterne. Das sogenannte Nutbuch*. The Carsten Niebuhr Institute of Ancient Eastern Studies, Denmark, 2007. ISBN 978-3-15-96137-8.

R. Manjula and R. Anitha. Identification of Encryption Algorithm Using Decision Tree. *CCSIT 2011, Part III, CCIS 133*, pages 237–246, New Delhi, India, 2011.

W. Mason. Bionsgadgets. January, 2021. https://bionsgadgets.appspot.com.

Beáta Megyesi, Bernhard Esslinger, Alicia Fornés, Nils Kopal, Benedek Láng, George Lasry, Karl de Leeuw, Eva Pettersson, Arno Wacker, and Michelle Waldispühl. Decryption of historical manuscripts: the DECRYPT project. *Cryptologia*, 2020. DOI:10.1080/01611194.2020.1716410.

M. Nuhn and K. Knight. Cipher Type Detection. *Conference on Empirical Methods In Natural Language Processing*, pages 1769–1773, Doha, Quatar, October, 2014.

A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barkerand S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. *Computer Security*. NIST - National Institute of Standards and Technology, April, 2010.

G. Sivagurunathan, V. Rajendran, and T. Purusothaman. Classification of Substitution Ciphers using Neural Networks. *IJCSNS International Journal of Computer Science and Network Security*, volume 10, pages 274–279, March, 2010.

C. Tan and Q. Ji. An Approach to Identifying Cryptographic Algorithm from Ciphertext. *8th IEEE International Conference on Communication Software and Networks*, Chengdu, Sichuan Province, China, 2016.

P. Tino, L. Benusova, and A. Sperduti. Natural-logarithm-rectified activation function in convolutional neural networks. China, December, 2019. IEEE 5th International Conference on Computer and Communications.

Z. Zhao, Y. Zhao, and F. Liu. The Research of Cryptosystem Recognition Based on Randomness Test's Return Value. *Cloud Computing and Security - 4th International Conference*, pages 1–15, Haikou, China, June, 2018.