Cryptography and Computer security
Classifying classical ciphers using random forest

Rok Ivanšek, IŠRM, II

May 27, 2017

Abstract

When encrypting a plaintext into a ciphertext, the idea is to hide the actual content of the underlying message, thus making it unreadable for a potential harmful adversary. Ciphertexts are therefore "gibberish" texts that do not show any structure that would resemble a natural language. Despite this fact, we can still find patterns in the ciphertexts. The patterns are clearer, when the underlying encryption rule is more transparent, like in the case of classical ciphers. In this report I show that we can use these patterns to match a set of ciphertexts to the corresponding cipher. By carefully engineering features, we can use a random forest technique to train a classifier model that is able to classify ciphertexts of classical ciphers.

1 Introduction and related work

The hardest part in deciphering a given ciphertext, is to find out what type of encryption was used to produce it. Once this information is known, deciphering becomes a lot easier. By knowing the algorithm that was used, we understand the complexity of the encryption and we can narrow down the methods we will use, to decipher the given ciphertext. Through the recent years statistical classification techniques proved to be extremely successful in a large set of domains, such as computer vision, speech recognition, handwriting recognition, natural language processing and many more. In this assignment we use statistical classification, in particular the ensemble method called random forest [1], to try and build a model that is able to distinguish between different classical ciphertexts. An accurate classification model would help a lot in the early stages of deciphering ciphertexts and may save countless hours of time for a cryptologist. In the domain of ciphertexts, classification has been attempted for classical substitution ciphers using a neural network by Sivagurunathan, Rajendran and Purusothaman [4], modern block ciphers like DES, 3DES and AES using support vector machines by Dileep and Chandra Sekhar[3]. Another approach was used by Maheshwari [8] for both classical and modern ciphers. In [9] Mishra and Bhattacharjya used a combination of different methods to classify modern block and stream ciphers and a clustering study was done for ciphers from the finalists of the AES contest in 2000 by Souza, Tomlinson, Figueiredo and Luiz [12]. In our study we follow a typical data science pipeline and we describe it in detail in each section of this report. First we generate training data by encrypting common plaintexts with different classical encryption techniques. We describe this process in section 2. Then we engineer features. We describe which features were used and the reason behind it in section 3. We do some simple exploratory data analysis and present the results in section 4. Next we train and tune the model and give a short description of the random forest technique in section 5. And in the end we present and interpret the results of the classification in section 6. In addition to the mentioned sections we describe the framework and comment on the work in section 7.

2 Data

In this section we describe the data. Where we obtained the plaintexts, how we processed them and which ciphers were chosen to produce the ciphertexts. Short descriptions of most of the used ciphers are also given.

2.1 Plaintexts

For plaintexts we randomly choose 1000 texts from the 20 Newsgroups dataset. We used the preprocessed dataset from Ana Cardoso Cachopo [2]. The preprocessing made on original texts from the 20 Newsgroups dataset was:

- substitute TAB, NEWLINE and RETURN characters by SPACE,
- keep only letters (that is, turn punctuation, numbers, etc. into SPACES),

- turn all letters to lowercase,
- substitute multiple SPACES by a single SPACE,
- the title/subject of each document is simply added in the beginning of the document's text.

In addition to the preprocessing already made, we take out the white spaces and trim the texts to length of 500 characters. This way we get 1000 strings of length 500, each representing a part of a randomly selected text from the dataset. This strings were the plaintexts used in the analysis.

2.2 Ciphertexts

To encrypt the plaintexts we use the python library pycipher [7]. To make the classification task as representative as possible, a variety of different ciphers is chosen. Some would produce similar ciphertexts, like the Permutation and Caesar cipher and some ciphertexts, that are clearly distinguishable from each other, like the ADFGVX and the Vigenere cipher. While encrypting, the key of the cipher was always chosen randomly out of all possible combinations, except where stated differently. Each plaintext was encrypted with every chosen cipher. This way we obtain 6000 cipherthexts, a 1000 for each of the chosen ciphers. The following ciphers were chosen.

In the description of the ciphers we always assume that $x = (x_1, x_2, ..., x_m)$ is a plaintext consisting of m integers from ring $\mathbb{Z}_{26}$ which map to letters in the english alphabet.

2.2.1 Affine cipher

The affine cipher is in essence a standard substitution cipher, meaning that each letter encrypts to one other letter. The key are integers a and b. The encryption rule is

$$e(x_i) = (ax_i + b) \mod 26, \forall i$$

where a is relatively prime to 26 and b is an arbitrary integer in $\mathbb{Z}_{26}$.

2.2.2 Vigenere cipher

The Vigenere cipher originates from the 16th century. It encrypts the ciphertexts in blocks of n characters. For a chosen keyword $K = (k_1, k_2, ..., k_n)$ of length $n < m$, it encrypts the plaintext following the rule

$$e(x_1, x_2, ..., x_m) = (x_1 + k_1, x_2 + k_2, ..., x_n + k_n, x_{n+1} + k_1, ..., x_{2n} + k_n, x_{2n+1} + k_1, ...),$$

where all + operations are done in $\mathbb{Z}_{26}$.

2.2.3 ADFGVX cipher

The ADFGVX was a field cipher used by the German Army during World War I. It uses a modified Polybius square and a single columnar transposition to encrypt the plaintext. The cipher can encrypt all letters of the alphabet and numbers from 0 to 9. It uses two keys. First one is used to build a Polybius square and the second one is used for the columnar transposition. The complete procedure of encryption is as follows:

- Using the first key, a $6X6$ square is filled with letters and numbers, each column and row represents one of the ciphertext letters A,D,F,G,V or X. This way each letter in the alphabet and the numbers 0 to 9 are mapped in a one to one correspondence to pairs of ciphertext letters. This way we get a ciphertext twice as long as the plaintext and only consisting of ciphertext letters.
- To further encrypt the ciphertext obtained in the first part, we use the second key. We write down the ciphertext in rows under the second key and than sort the columns of the obtained table, so that the letters in the second key are in alphabetic order. Following this, we write down the letters from the sorted table column by column, to obtain the final ciphertext.

The important properties of the ADFGVX cipher are that it uses only the letters A,D,F,G,V and X in the ciphertexts and that the ciphertext is two times as long as the plaintext. You can read more about this cipher in [6].

2.2.4 Caesar cipher

The Caesar cipher is one of the earliest known and simplest ciphers. It is a shift cipher that shifts all the letters in the plaintexts by a key value k. It can be viewed a special case of the Vigenere cipher, where the key is of length 1. The encryption rule is

$$e(x_i) = (x + k) \pmod{26}, \forall i.$$

2.2.5 Permutation cipher

Also called the simple substitution cipher, it has been in use for many hundreds of years. It basically consists of substituting every plaintext character for a different ciphertext character. It differs from the Caesar cipher in that the cipher alphabet is not simply the alphabet shifted, it is completely jumbled. The key for this cipher is simply a jumbled alphabet. For example a key $k = AJPCZWRLFBDKOTYUQGENHXMIVS$ would impose the following encryption rule

$$ABCDEFGHIJKLMNOPQRSTUVWXYZ \rightarrow AJPCZWRLFBDKOTYUQGENHXMIVS,$$

meaning a letter in the left string would encrypt into a letter in the right string that is in the same place. For example M would encrypt to T.

2.2.6 Playfair cipher

The Playfair cipher is a digraph substitution cipher from the 19th century that was also used in the World War I by the British forces. It encrypts pairs (digraphs) of letters using a key square. The full description of the encryption is too long to include in this report. Read more about this cipher in [6].

3 Feature engineering

In order to express patterns in the ciphertexts we have to come up with interesting features. A feature is in essence any property of the ciphertexts that can be expressed in a qualitative or a quantitative way. The features that we decided to use are closely related to the properties of the chosen ciphers. They are mostly quantitative measurements that are used, when one attempts to break the chosen ciphers.

3.1 Distribution of letters

The first most simple group of features one can extract is the distribution of letters. This just means that we count, how many times each letter appears in the ciphertext. This way we obtain 26 features, one for each letter in the alphabet. Using only this simple group of features we would already be able to distinguish between some of the ciphers that use only a subset of letters from the alphabet, like the ADFGVX from other ciphers that use the whole alphabet.

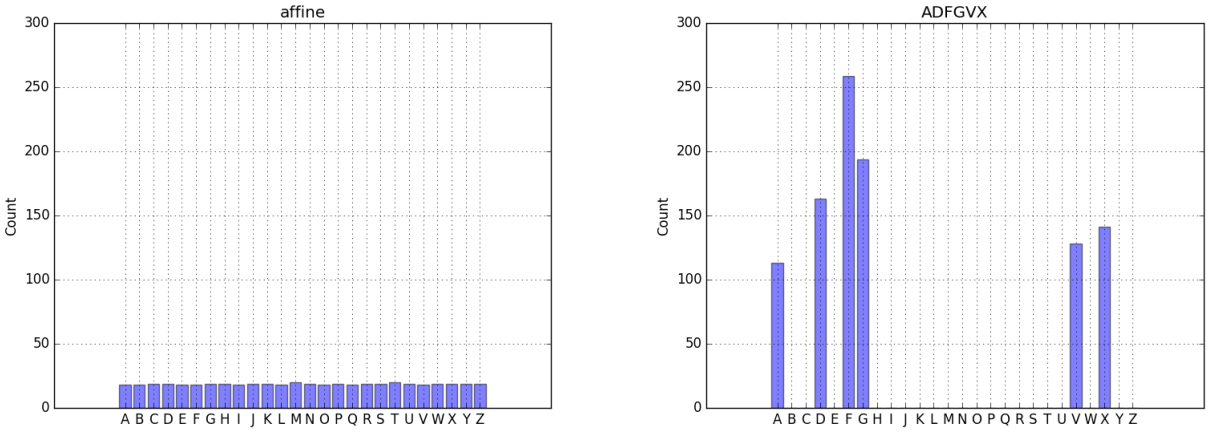

Figure 1: Average distribution of letters in the ciphertexts for the Affine cipher (left) and the ADFGVX cipher (right).

3.2 Adjacent duplicates

Another simple feature that proved to be efficient in a similar ciphertexts classification task [4] is the number of adjacent duplicates. It is particular useful when recognizing the Playfair cipher, since this cipher should have a substantially lower number of adjacent duplicates than other ciphers, due to its encryption rule.

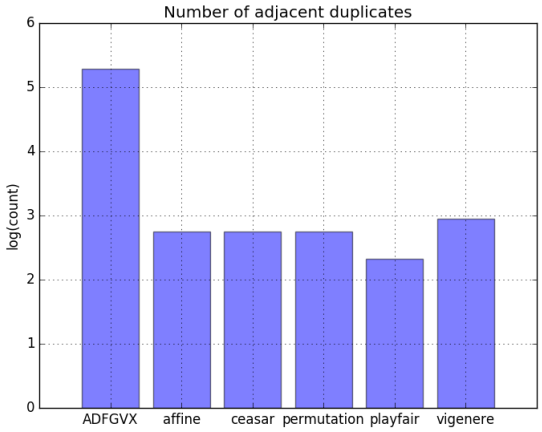

Figure 2: Logarithm of the average number of adjacent duplicates for the used ciphers.

3.3 Repeating bigrams

In a classical text classification an n-gram (usually a bigram) is a sequence of n adjacent tokens in a text. Tokens can be words, syllables or letters. Using bigrams of letters I extracted two features. First was the number of unique bigrams that repeat at least once in a ciphertext and second was the frequency of the most frequent bigram in a ciphertext.

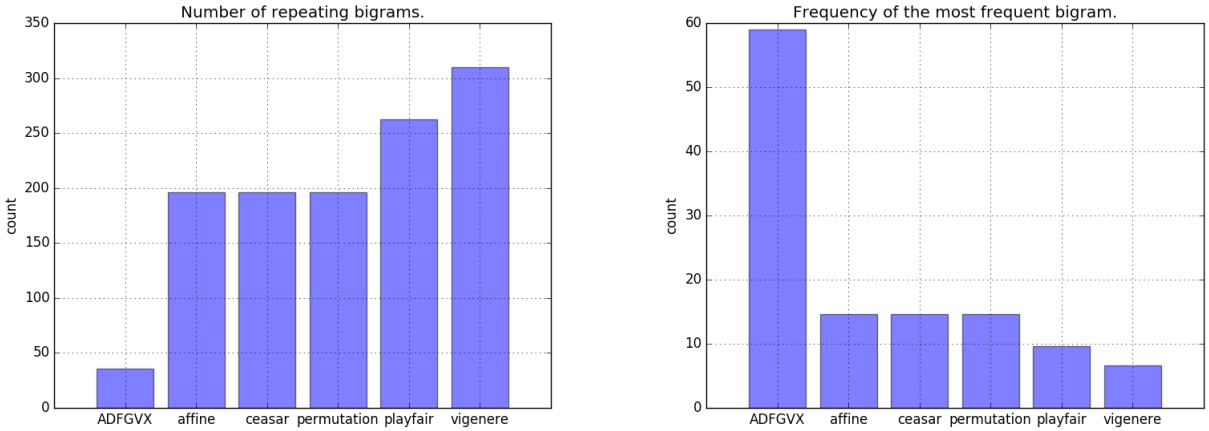

Figure 3: Average number of unique repeating bigrams (left) and the average frequency of the most appearing bigram (right) in the ciphertexts.

3.4 Index of coincidence

The index of coincidence (IC) along with Kasiski's test is used in breaking the Vigenere cipher. It tells us the probability that we will get two matching letters, if we randomly select two letters from a given text. The formula for calculating the IC for a text x is

$$\mathrm{IC}(x) = \sum_{i=0}^{25} \frac{f_i(f_i - 1)}{d(d-1)},$$

where f_i are the frequencies of letters in x and d is the length of x.

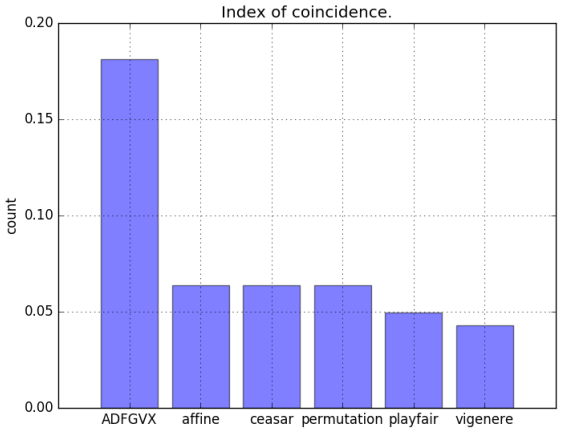

Figure 4: Average index of coincidence.

The IC of an English plaintext is approximately 0.065. The Affine, Caesar and Permutation ciphers are examples of single letter substitution ciphers, where the mapping that determines the encryption of letters is bijective (one-to-one). It is therefore no surprise that the IC of their ciphertexts is the same as the IC of the English plaintexts.

4 Exploratory data analysis

One common step before moving on to actual training of the model, is to draw some rough graphs representing the data that will be used for training. This is to see how features interact with each other and to get an idea of how the data clusters.

Here I present just one graph to show that the chosen features separate the data in a clear way. I use a subsample of 10% of the data. The features were scaled zero mean and unit variance, and then multidimensional scaling was used to shrink down the data to two dimensions.

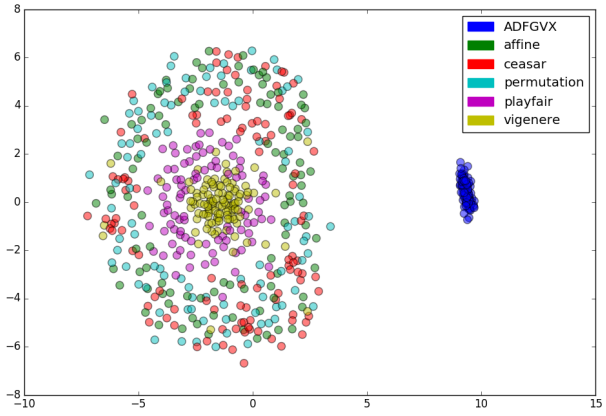

Figure 5: MDS(multidimensional scaling) used on a subsample of the data.

In figure 5 we can see how the ADFGVX cipher is clearly separated from the others. The Vigenere and the Playfair appearing in the inner circles are also separable from the other ciphers, while the Affine, the Caesar and the Permutation ciphers all lay mixed together in the outer most circle. This is an encouraging sight. It tells us that the classifier should produce good results.

5 Random forest classifier

Random forest is an ensemble method. It works by building multiple simple decision trees on different subsamples of the data and than combining the results to produce a stronger model. It can be used for classification

or regression. We will not go into detail about the workings of the decision tree and random forest techniques, since this is a broad topic and it is not the main focus of this report. We advise the reader to learn more in [1]. For now it suffices to say that this is a robust machine learning technique that works well on different domains in data science. The main parameter of the random forest is the number of decision trees built. It generally holds that more is better and we usually build as much trees as our computing capabilities allow us, however the accuracy of the model is expected to stop improving substantially at some point.

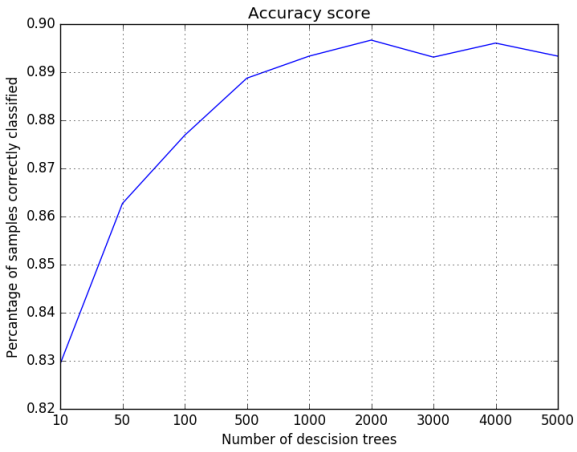

Figure 6: Determining the optimal number of decision trees in the random forest.

On graph 6 we can see how the accuracy score of the random forest changes as we increase the number of decision trees in the forest. We can see that the accuracy of the model does not change significantly for more that 2000 decision trees. The score was calculated on the training set using a 5-fold cross validation.

6 Results

Training a random forest on the train set of the data, building 2000 decision trees, I obtained an average accuracy of around 92% on the test set.

	precision	recall	f1-score	support
ADFGVX	1.00	1.00	1.00	195
Affine	0.73	0.77	0.75	182
Caesar	0.91	0.99	0.95	212
Permutation	0.83	0.75	0.79	210
Playfair	1.00	1.00	1.00	208
Vigenere	1.00	0.95	0.98	193
avg/total	0.91	0.91	0.91	1200

Table 1: Classification report showing the main classification metrics.

Table 1 shows some of the more common classification metrics for one specific run on the random forest classifier. Precision tells us the fraction of samples classified as some class that actually belong to this class, while recall or sensitivity tells us the fraction of samples from some class that were classified to be in this class. Example: All of the samples predicted to be Vigenere, were in fact Vigenere so the precision for the Vigenere class is 1.00, however 9 of the Vigenere samples were predicted to be Caesar so the recall is only 0.95. The $f1$-score is the harmonic mean of precision and recall, calculated with the formula

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

and support simply tells us how many samples of the class were in the test set. Table 2 shows the confusion matrix of the classification.

	ADFGVX	Affine	Caesar	Permutation	Playfair	Vigenere
ADFGVX	195	0	0	0	0	0
Affine	0	141	11	30	0	0
Caesar	0	1	210	1	0	0
Permutation	0	44	0	166	0	0
Playfair	0	0	0	0	208	0
Vigenere	0	0	9	0	0	184

Table 2: Confusion matrix of the classification.

We can see that the samples of ADFGVX and Playfair ciphers were all correctly classified and only 9 out of the 196 samples of the Vigenere cipher were classified as a Caesar cipher. The classifier however had some problems with the classification of the Affine, Caesar and Permutation ciphers. Out of 182 samples of the Affine cipher in the test set, 11 were classified as Caesar and 30 as Permutation. Out of 212 samples of the Caesar cipher, 1 sample was classified as Affine and 1 as Permutation. Lastly out of 210 samples of the Permutation cipher, 44 were classified as Affine.

7 Framework and Conclusions

For this project I used the Python programming language. I used Pycipher library [7] to encrypt the plaintexts and Scikit-Learn library [10] to sample the data, train/tune the model and interpret the results. For the generation of graphs I used the Matplotlib library [5]. For the description of the ciphers I relied on lecture notes from the subject Cryptography and Computer Security taught by prof. Jurišič at the Faculty of Informatics, practical cryptography website [6] and Stinson's book [11].

The classification results are encouraging. Using a limited number of fairly simple features we are able to build a model that exhibits an average classification score higher that 90% and it is able to distinguish even between ciphers that use similar encryption rules. To go further and attempt classification of modern ciphers like DES and AES one has to approach the problem with a bit more care. The patterns there are not so clearly visible and by using simple features like the distribution of letters we are not able to distinguish between modern ciphers. Instead more advanced techniques like the ones used in [4] and [12] should be employed.

References

[1] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] A. Cardoso-Cachopo. Improving Methods for Single-label Text Categorization. PdD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

[3] A. D. Dileep and C. Chandra Sekhar. Identification of block ciphers using support vector machines. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 2696–2701. IEEE, 2006.

[4] V. Rajendran G. Sivagurunathan and T. Purusothaman. Classification of substitution ciphers using neural networks. *International Journal of computer science and network Security*, 10(3):274–279, 2010.

[5] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[6] J. Lyons. Practical cryptography. `http://practicalcryptography.com/`. Accessed: 2017.

[7] J. Lyons. pycipher. `https://github.com/jameslyons/pycipher`, 2016.

[8] P. Maheshwari. *Classification of ciphers*. PhD thesis, Indian Institute of Technology, Kanpur, 2001.

[9] S. Mishra and A. Bhattacharjya. Pattern analysis of cipher text: A combined approach. In *Recent Trends in Information Technology (ICRTIT), 2013 International Conference on*, pages 393–398. IEEE, 2013.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] D.R. Stinson. *Cryptography: theory and practice*. CRC press, 2005.

[12] A. Tomlinson WAR de Souza and Luiz MS LMD de Figueiredo. Cipher identification with a neural network. 2012.