1 Probability Theory Review

1.1 Probabilistic Independence

a) With rolling the dice twice you have 36 different outcomes. Event A gets fullfilled in 3 cases. 12 cases fullfil event B and C. Event D gets fullfilled in 15 cases. And 22 cases for Event E. This results in following Propabilties for the different events:

$$P(A) = \frac{3}{36} = \frac{1}{12}$$
$$P(B) = \frac{12}{36} = \frac{1}{3}$$
$$P(C) = \frac{12}{36} = \frac{1}{3}$$
$$P(D) = \frac{15}{36} = \frac{5}{12}$$
$$P(E) = \frac{22}{36} = \frac{11}{18}$$

b) Event A and B get satisfied in the cases (6,5),(5,6),(6,6) to show dependece we have to show following equation.

$$P(A \cap B) \neq P(A)P(B)$$

$$P(A \cap B) = \frac{3}{36} \neq \frac{1}{36} = \frac{3}{36} * \frac{1}{3} = P(A) * P(B)$$

since both sites are not equal we know that A is dependent of event B.

c) $A \cap C$ can not be fullfilled. And since $P(A) > 0$ and $P(C) > 0$ we know that

$$P(A \cap C) = 0 \neq P(A)P(C) > 0$$

Which shows that A is depented of event C.

d) $D \cap E$ gets satisfied by the cases (1,2),(2,3),(3,4),(4,5),(5,6) which implies $P(D \cap E) = \frac{5}{36}$ Since

$$P(D \cap E) = \frac{5}{36} \neq \frac{55}{216} = \frac{5}{12} * \frac{11}{18} = P(D)P(E)$$

we know that D and E are dependend.

1.2 Communication through a Noisy Channel

a) For a random variable of source X, the probability mass function can be denoted as:

$$P_X(x_k) = \ P(X = x_k),\ \ for\, k = 0, 1$$

So for the source symbol 0

$$P_X(0) = P(X = 0) = p$$

and for 1

$$P_X(1) = P(X = 1) = 1 - p$$

b) Y is the random variable of the receiver symbol which can have the value 0 and 1. So the mass function of Y given X which implies the channel's error probability is as follows:

$$P_Y(y_k) = \ P(Y = y_k),\ \ for\, k = 0, 1$$

$$P_Y(0 \mid (X = x_k = 1)) = \epsilon_1$$

$$P_Y(1 \mid (X = x_k = 0)) = \epsilon_0$$

c) The probability of transmitting the symbol sequence "1001110" is:

$$(1 - p)^4.p^3$$

d) Let, c denotes receiving the signal correctly. The probability of randomly chosen symbol which is received correctly is:

$$P(c) = P(c \mid 0).P(0) + P(c \mid 1).P(1)$$

$$= p.(1 - \epsilon_0) + (1 - p).(1 - \epsilon_1)$$

e) The probability of receiving the transmitted symbol sequence 1011 correctly is:

$$P(C) = P(1\ \ C)^3.P(0\ \ C), where\, C\ =\ receiving\, correctly$$

$$= (1 - \epsilon_1)^3.(1 - \epsilon_0)$$

f) The probability of receiving "1101" is as follows: [R=Receiving, T=Transmitting]

$$P(R\,1101) = (P(R1 \mid T1).P(T1)+$$

$$P(R1 \mid T0).P(T0))^3.P(R0 \mid T1).P(T1) + P(R0 \mid T0).P(T0))$$

$$= ((1 - \epsilon_1).(1 - p) + \epsilon_0.p)^3.(\epsilon_0.(1 - p) + (1 - \epsilon_0).p)$$

g) Probability of receiving a transmitted O[000] correctly after introducing redundancy:

$$P(000, 001, 010, 100 \mid 000)$$
$$= P(000 \mid 000) + P(001 \mid 000) + P(010 \mid 000) + P(100 \mid 000)$$
$$= (1 - \epsilon_0)^3 + 3.(1 - \epsilon_0)^2.\epsilon_0$$

h) The probability of transmitting 0[000] that the received signal is 101:

$$P(000 \mid 101) = \frac{P(000 \wedge 101)}{P(101)}$$

$$= \frac{P(101 \mid 000).P(000)}{P(101 \mid 000).P(000) + P(101 \mid 111).P(111)}$$

$$\frac{\epsilon_0^2(1 - \epsilon_0).p}{\epsilon_0^2(1 - \epsilon_0).p + (1 - \epsilon_1)^2.\epsilon_1.(1 - p)}$$

i) The noisy channel can be a good model of human communication with natural languages. It actually removes the part of guessing by getting close to the optimal channel capacity. Evaluating the results of the design assists to develop the system even more. By applying some variants like the question g, it can be designed such a way that the optimal performance is more achievable. It has its impact on Handwriting recognition, Text Generation, Machine Translation and so on.

2 Character N-grams and Entropy

2.1 N-grams Frequency Analysis

<u>Exercise 2.1: N-grams Frequency Analysis</u>

In the following part of work we collected n-gram frequency counts for German and English text corpora.

a) Word tokens of each text corpus were converted into a set of n-grams (up to n = 4). For this purpose a method *char_ngrams* was created, it takes as an input a word token and and an integer m and returns a list of all possible n-grams in the token from n = 1 up to n = m.

b) Then we generated a table of the top 15 frequent character unigrams (1-grams), bigrams (2-grams), trigrams (3-grams), and 4-grams for each text corpus. The most common n-grams of text corpora are represented in Table 1 for English and German.

Table 1. The top 15 frequent character n-grams in English and German.

	English				German			
	Uni-grams	Bi-grams	Tri-grams	Four-grams	Uni-grams	Bi-grams	Tri-grams	Four-grams
1	e	th	the	tion	e	en	der	sche
2	t	he	ion	atio	n	er	ung	chen
3	i	in	tio	ment	r	de	sch	isch
4	o	on	ing	sion	i	ch	die	eine
5	n	ti	and	emen	s	un	ich	lich
6	a	an	ati	comm	t	te	che	icht
7	r	re	ent	arti	a	ei	ein	rung
8	s	er	men	ions	d	ie	gen	ngen
9	c	at	for	with	u	ge	und	iche

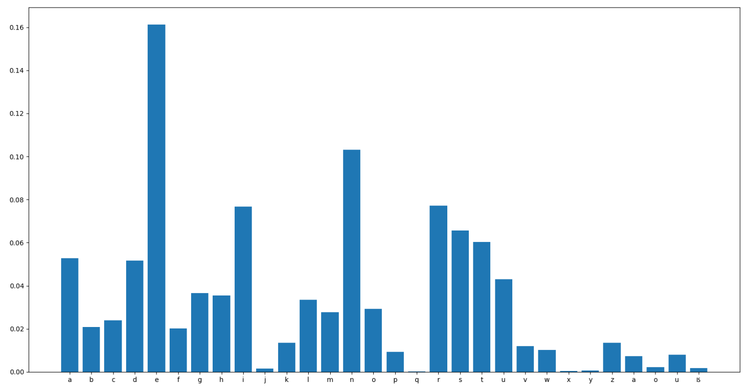
10	h	io	pro	hall	g	in	den	ungs
11	l	of	ate	shal	h	ng	ten	unge
12	d	or	com	rtic	l	es	ver	über
13	u	en	con	ting	o	nd	nde	komm
14	m	nt	ter	that	m	be	hen	ende
15	p	es	sio	ther	c	st	cht	nder

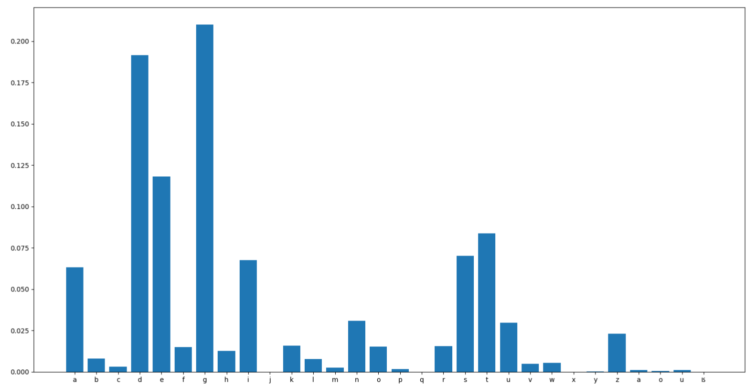
c) Assuming that the identity of the languages and the original word forms in the two corpora are unknown, it is possible to identify the language based on the distribution of frequent n-grams. According to the Zipf's law words that are most frequently used tend to be shorter. In this case we could suppose that some of the n-grams from the top-list are represented by words, most frequently used in English or German. Furthermore, other n-grams from the list can be recognized as prefixes and suffixes, which are characteristic for the languages.

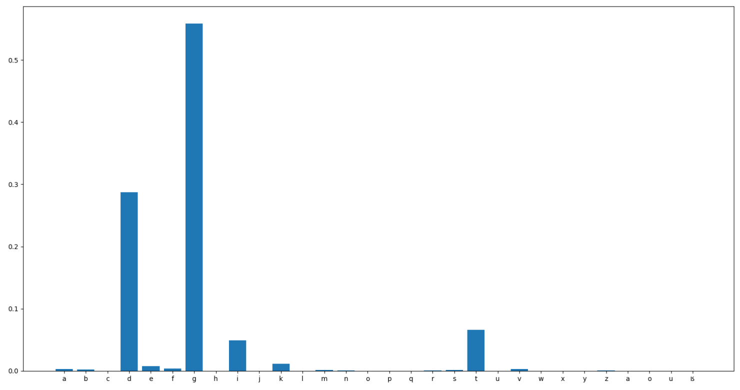
2.2 N-gram Probability Distributions

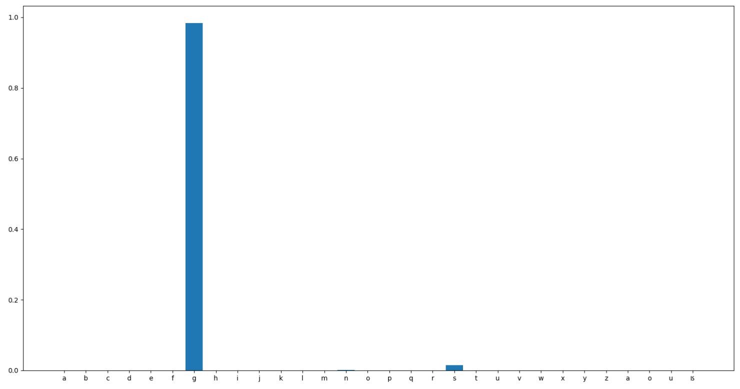
a) see prob_dist.py We gave out the probability for an empty history if for the given historie no possible following character was found in the n-grams to ensure that the sum of the probability distribution always sums up to one. This was tested with assert and different histories.

b) Below are the plots for the probability distribution for the histories ‘’, ‘n’, ‘un’ and ‘gun’. We can observe that with a growing history we get less and less possible characters based on the n-grams collected on the sample. In the first case with an empty history, every character is possible even if some are more probable than others. With ‘n’ as history, some of the characters get eliminated. With the biggest history ‘gun’ we can see that there are only two possible characters that can follow: g and s, where g has a high probability since it was more frequent in the sample.









c) see prob_dist.py

We can see that the entropy grows to zero when the history grows, which

	Entropy
‘’	2.89886
‘n’	2.4599
‘un’	1.22248
‘gun’	0.09247

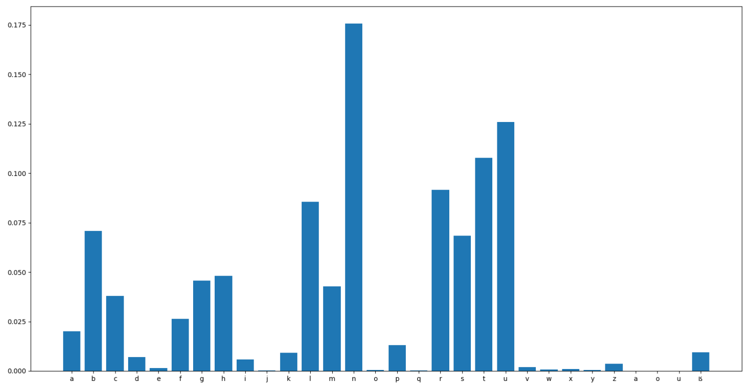
is based on the fact that the probabilities are distributed on less characters with the minimum entropy 0 at one character with 100%probability.

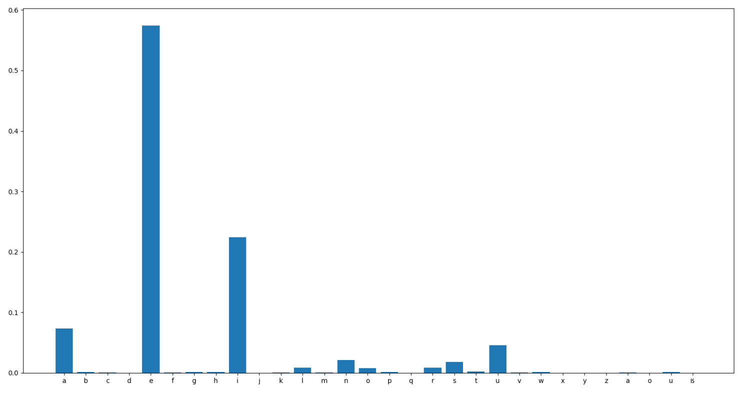
d) Based on the observation in c) we know that the maximum entropy we be achieved if the probability would be evenly distributed among all characters, which would in the german alphabet result in the entropy:

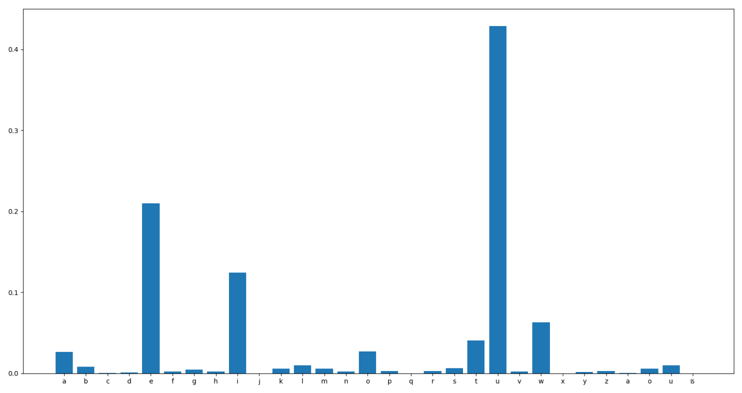
$$30 * \frac{1}{30} * log(\frac{1}{30}) = log(\frac{1}{30}) = 1.47712125472$$

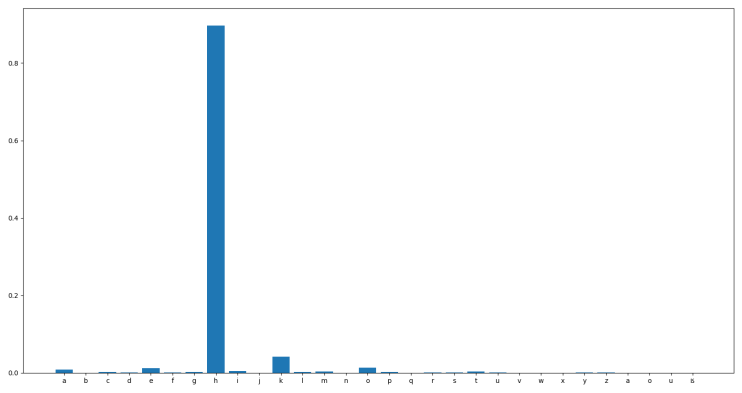
	Entropy
‘a’	2.59749
‘d’	1.38149
‘z’	1.86678
‘c’	0.55237

e) Below we have the plots for the histories ‘a’, ‘d’, ‘z’ and ‘c’. This shows that not only the size of the historie is important for the probability distribution like we saw in part b). Even when the history has only one single character the distribution and with that the entropy can change a lot. The vocal a enables a lot of possible following characters based on the n-grams, which leads to a high entropy. d and z can lead to all the different letters in the alphabet but only 3 to 4 are realy frequent in the n-grams. This leads to a smaller entropy than for example the character a. The history c has a high probability to be followed by h, since ‘ch’ is one combination often used in the german language. Since it is only one case with a realy high probability the entropy is quite small.




0.6
0.5
0.4
0.3
0.2
0.1
0.0
a b c d e f g h i j k l m n o p q r s t u v w x y z a o u ß


0.4
0.3
0.2
0.1
0.0
a b c d e f g h i j k l m n o p q r s t u v w x y z a o u ß


0.8
0.6
0.4
0.2
0.0
a b c d e f g h i j k l m n o p q r s t u v w x y z a o u ß

2.3 Cryptoanalysis

Exercise 2.3: Cryptographic Analysis

In the following part of work we approach the n-gram distributions of English and German to crack the substitution cipher. The algorithm provides deciphering based on bigrams and trigrams distributions that were produced in *frequency_analysis.py*.

The substitution key: **nqjcgxyszuhvkboamitpefwlrd**

Explanation of the algorithm:

1. The algorithm starts with generating a random key, and incrementally improve it changing two chars in the step. Then the algorithm restarts. The number of restarts and steps in each restart is preliminarily defined.
2. **Improving the key.** The keys is judged based on the decryption. A new key is chosen closely to the previous key based on bigrams distribution by *neighboring_keys()*. Decrypted text is converted to 2-grams and yields keys by the following: one letter in each 2-gram is swapped to random letter so that a new bigram had higher probability.
3. **Optimization.** The local maximum of the fitness function refers to the most appropriate key among those produced in each restart. Local maximums in each restart are searching in the following way: if the value of the fitness function is better than in the previous step, the previous memorizes key is changed to the current one. This optimization is provided by *steepest_ascent()*.
4. **Decryption evaluation.** A decryption is evaluated based on trigrams distribution by *trigram_string_prob()*. The decryption is converted to 3-grams and the method calculates a value - sum of 3-grams' probabilities (more precisely, logarithms' of the probabilities). The larger the value, the better decryption. These values are the fitness values for *steepest_ascent()*.

5. **Final evaluation.** In *crack_ciphertext()* the local maximums are compared to each other, and the global maximum refers to the key which is returned by the method and used to decipher text.
6. Finally, decryptions for both languages are compared and the best one is chosen according to larger value of the fitness function.