Project 2 - Data compression and languages

Luke Roantree

December 30, 2018

Abstract

In this report the efficiency of encoding data from different written languages was investigated, with a focus on Huffman Codes. Construction of Huffman Codes was described and, using a freely available source for Letter Frequencies, efficiencies of encoding large and small texts in different languages were compared. The benefit of ad-hoc code generation was investigated, and compared to the previous encodings. It was found that binary Huffman Codes for all languages had an average word length less than 1% above their binary entropies, matching expectations due to Shannon's compression limit and Huffman Code's optimality. It was found that when encoding the sample used to generate letter frequencies, there was a slight decrease obtained in average word length. d-ary codes were further investigated, and their use cases considered, as was the impact of using text samples with differing contexts in generating letter frequencies; on average, $35 - 45\%$ differences in letter frequencies were obtained for the same languages between using text samples from Twitter, and from the Universal Declaration of Human Rights.

1 Background of Data Compression & Huffman Encoding

Data compression is used in many scenarios from audio, to image, to text. Each use case has different requirements for the compression; for example, with plaintext messages uniquely decipherable, lossless codes are usually required, while for images lossy compression can be used to achieve a much smaller file size, and still decode to a recognizable image. With audio, in the case of streaming, instantaneous codes are required.

Notable uses of data encoding include Braille - where written text information is encoded in physical bumps, and Morse code - where written text is Huffman encoded in two lengths of pulse allowing near-instant transmission via electrical cables.

Shannon defines the entropy of a source as

$$\text{Entropy} = -\sum_{p_i} p_i \log p_i \tag{1}$$

Where p_i are the probabilities of occurrence associated with the letters in the source, and further claims that the entropy of a source is the limit of compressibility.

2 Constructing Huffman Codes

2.1 Investigating Letter Frequencies

To generate Huffman Codes for letters in different languages, one needs to know the probabilities of each letter occuring for each language. I initially use a table available on wikipedia - `https://en.wikipedia.org/wiki/Letter_frequency` - that contains Letter Frequency estimates for many languages.

This table is not completely usable as-is for several reasons including that several cells contain characters such as *, $\sim$, etc. denoting clarifications/justifications about the given probability - I choose to handle these by simply removing the characters.

Additionally, in order to more easily compare the languages, I choose to initially discard the probabilities of non english alphabet characters and re-normalise the probabilities of the remaining ones - this is equivalent to considering the efficiency of encoding *only* the english alphabet for each language, while if the probabilities were not re-normalised it would be equivalent to estimating the efficiencies for encoding *all* the symbols used in each alphabet. The main reasoning for choosing to re-normalise in this case is that the wikipedia tables' columns for each language do not sum to 100% initially; potentially leading to incorrect calculations.

Figure 1 (below) shows the cleaned data from Wikipedia, where we can see that, for the specified languages, 'e', 't', and 'a' consistently have high probabilities of occuring, while 'n' and 'o' are very common in English, French, and Italian; but far less common in German.

Figure 1: Table of Letter Frequencies from cleaned Wikipedia data

Letter	English	French	Italian	German
a	0.081671	0.078537	0.118743	0.066708
b	0.014920	0.009267	0.009372	0.019308
c	0.027820	0.033529	0.045506	0.027969
d	0.042530	0.037736	0.037771	0.051966
e	0.127021	0.151345	0.119218	0.167856
f	0.022280	0.010964	0.011657	0.016953
g	0.020150	0.008907	0.016621	0.030805
h	0.060941	0.007580	0.006430	0.046858
i	0.069661	0.077437	0.102547	0.067056
j	0.001530	0.006305	0.000111	0.002744
k	0.007720	0.000761	0.000091	0.014507
l	0.040250	0.056116	0.065817	0.035187
m	0.024060	0.030526	0.025397	0.025942
n	0.067491	0.072973	0.069588	0.100083
o	0.075071	0.059612	0.099402	0.026556
p	0.019290	0.025929	0.030896	0.006859
q	0.000950	0.014008	0.005106	0.000184
r	0.059871	0.068838	0.064371	0.071694
s	0.063271	0.081746	0.050358	0.074427
t	0.090561	0.074505	0.056849	0.063002
u	0.027580	0.064909	0.030442	0.042650
v	0.009780	0.018904	0.021201	0.008661
w	0.023600	0.000504	0.000334	0.019666
x	0.001500	0.004392	0.000030	0.000348
y	0.019740	0.001316	0.000202	0.000399
z	0.000740	0.003353	0.011940	0.011609

The code used to obtain and clean the data from Wikipedia is contained in file 'GenerateCSVLetterFrequencies.ipynb', attached.

With the data in Figure 1, and Shannon's definition of entropy;

$$\text{Entropy} = -\sum_{p_i} p_i \log_2 p_i \tag{2}$$

The binary entropies for the given languages were calculated using 'calculate_binary_entropies()' from file 'Project2Tools.py';

Figure 2: Table of Binary Entropies

Language	Binary Entropy
English	4.175787
French	4.043450
Italian	3.981666
German	4.083792

Figure 2 shows that Italian should theoretically be able to be encoded the most efficiently, while English should be the least efficient to encode

2.2 Recursive Generation of Binary Huffman Codes

To generate binary Huffman Codes using the probabilities from Figure 1, I first define a some terminology;

- 'leaf nodes' will represent symbols/symbol combinations that will have codes assigned to them.
- 'priority order' will represent the leaf_nodes sorted by their corresponding probabilities

The method used by 'recursive_huffman' from 'Project2Tools.py' for generating the binary Huffman Codes is described by the following steps;

- Define base case - when only 2 elements are in leaf_nodes, they are taken to be the root; assign them '0' and '1'
- assign variable 'lfn' a copy of the current leaf_nodes
- find the 2 nodes with lowest probability (say A,B)
- combine their symbols to create new node (say AB) with asscociated probability of prob(A) + prob(B)
- delete nodes A,B from lfn
- (recursion step) call recursive_huffman() with the newly updated leaf_nodes
- once the base case has been reached, and results propagated back to this level, define code_so_far as the longest HC assigned so far.
- assign the codes for the 2 least probable nodes to be this longest code so far, appended with '0' and '1'
- return the codes

2.3 Huffman Codes for Several Languages

Figure 3: Table of Huffman Codes

Letter	English	French	Italian	German
a	1110	1111	011	1001
b	110000	1001101	1100110	110110
c	01001	10010	11110	01101
d	11111	0010	11100	0100
e	100	110	100	111
f	00101	001101	1100111	110100
g	110011	1001100	110010	10000
h	0110	0101001	11101111	0001
i	1011	1110	001	1010
j	001001011	0101000	111011100110	110101101
k	0010011	0011001001	1110111001111	010111
l	11110	0100	1011	10001
m	00111	01011	01000	01010
n	1010	1010	1101	001
o	1101	0110	000	01100
p	110001	00111	11000	11010111
q	001001001	010101	111011101	11010110010
r	0101	1000	1010	1011
s	0111	000	11111	1100
t	000	1011	0101	0111
u	01000	0111	01001	0000
v	001000	100111	111010	1101010
w	00110	0011001000	1110111000	110111
x	001001010	0011000	1110111001110	11010110011
y	110010	001100101	11101110010	1101011000
z	001001000	00110011	1110110	010110

Figure 3 shows the Huffman Codes for alphabets based on the data from Figure 1; as expected, the codes for letters that are most frequently used such as 'e', 't', and 'a' are very short, while the codes for infrequently used letters such as 'x' and 'z' are comparatively very long.

Figure 4: Table of Average Word Length of Huffman Codes and Binary Entropy Theoretical Limits

Language	Average Word Length	Binary Entropy	Relative Difference (%)
English	4.205062	4.175787	0.701064
French	4.081581	4.043450	0.943050
Italian	4.011657	3.981666	0.753215
German	4.115869	4.083792	0.785486

Figure 4 describes the efficiency of the Huffman Codes from Figure 3, by comparing the average word length (weighted by probability of occuring) for each language's codes with the binary entropy of that language, which, by **INSERT**'s theorem, is the theoretical minimum average word length that is possible to obtain. It can be seen that each AWL is greater than the binary entropy as expected, and also that each is very close to optimal, all less than 1% over the binary entropy.

3 Measuring Effectiveness of Huffman Encoding

3.1 Comparison With Trivial Block Code

Figure 5: Table of Trivial Block Codes (Laguage Independent)

Letter	Block_Code
a	00000
b	00001
c	00010
d	00011
e	00100
f	00101
g	00110
h	00111
i	01000
j	01001
k	01010
l	01011
m	01100
n	01101
o	01110
p	01111
q	10000
r	10001
s	10010
t	10011
u	10100
v	10101
w	10110
x	10111
y	11000
z	11001

Figure 5 shows a trivial block code implementation, made by simply assigning each letter a number, e.g. a$\rightarrow$ 1, b$\rightarrow$ 2, etc. and then converting each number to binary representation and left-padding them with 0s to be equally sized blocks.

It is clear to see that these block codes will have an average word length of exactly 5, roughly $24 - 26\%$ more than the binary entropies of each language. There can still be cases where a block code is more useful though, as using a block code doesn't require any prior knowledge of letter frequency, and it allows trivial, exact, calculation of encoded message length.

3.2 Code Lengths for Translations of One Sentence

Code lengths were investigated for encodings of translations of a single sentence, shown below;

- English: It is raining cats and dogs
- French: Il pleut des chiens et des chats
- Italian: Sta piovendo cani e gatti
- German: Es regnet Hunde und Katzen

The orginal phrase lengths (discounting spaces), total code lengths, ratios between the two lengths, and the codes themselves are shown below;

- Original Phrase Length: 22, Encoded Length English: 93, Ratio: 0.236559x
  encoded: ['1011', '000', '1011', '0111', '0101', '1110', '1011', '1010', '1011', '1010', '110011', '01001', '1110', '000', '0111', '1110', '1010', '11111', '11111', '1101', '110011', '0111']

- Original Phrase Length: 26, Encoded Length French: 104, Ratio: 0.25x
  encoded: ['1110', '0100', '00111', '0100', '110', '0111', '1011', '0010', '110', '000', '10010', '0101001', '1110', '110', '1010', '000', '110', '1011', '0010', '110', '000', '10010', '0101001', '1111', '1011', '000']

- Original Phrase Length: 21, Encoded Length Italian: 82, Ratio: 0.256098x
  encoded: ['11111', '0101', '011', '11000', '001', '000', '111010', '100', '1101', '11100', '000', '11110', '011', '1101', '001', '100', '110010', '011', '0101', '0101', '001']

- Original Phrase Length: 22, Encoded Length German: 84, Ratio: 0.261905x
  encoded: ['111', '1100', '1011', '111', '10000', '001', '111', '0111', '0001', '0000', '001', '0100', '111', '0000', '001', '0100', '010111', '1001', '0111', '010110', '111', '001']

It can be seen that while the encoding of the Italian translation has the shortest code length, the German translation has the best ratio of symbols used to encode to letters in the phrase; this means that, in this case, the letter frequency distributions for German are more optimal for encoding its phrase than the others.

However, the Italian letter frequency distribution paired with the Italian translation of the underlying information is more optimal to overall convert the information, that it is raining cats and dogs, to a numerical code.

3.3 Code Lengths for Translations of Large Text Sample

The previous comparison between the different Huffman Codes is a relatively poor representation of their general efficiency, as the sample to be encoded is so small; for example, the samples were between 21 and 26 characters long - comparing these lengths with the length of the alphabet, 26 characters, it is very unlikely that the letter frequencies in the text sample closely resemble the letter frequencies from the Wikipedia data. Additionally, it can be seen that the ratios in all cases show effective average word length being below the binary entropies and the AWLs of the Huffman Codes - this is because the phrases are so short that less common letters such as 'x' and 'z' simply did not occur.

A more appropriate comparison would require a much larger text sample. For this purpose, translations of the Universal Declaration of Human Rights (UDHR) were used. Using this text sample, the original sample lengths, the encoded lengths, and the ratios between them were as shown below;

- English → Original Length: 8673, Encoded Length: 36156, Ratio: 4.168800

- French → Original Length: 9626, Encoded Length: 38309, Ratio: 3.979742

- Italian → Original Length: 10459, Encoded Length: 41464, Ratio: 3.964433

- German → Original Length: 10102, Encoded Length: 41079, Ratio: 4.066422

It can be seen that the ratios produced by this sample much more closely represents the predicted AWLs of the Huffman Codes, differing by under 5% in all cases.

4 Using Ad-Hoc Code Generation

The effectiveness of using ad-hoc code generation was investigated, as it would ensure the letter frequencies used to make the codes would exactly match that of the sample to be encoded.

Downsides of using this method include that, if the encoded data is communicated, the recieving party would not be able to decode the data without also being given the letter frequencies or codes seperately. Additionally, if there are many messages to send, or speed of transmission is an issue, the calculation of the letter frequencies each time can lead to substantial increases in the length of time needed to encode them.

Conditions where it could be more useful include where the data to be transmitted consists of infrequent, long, messages such that the additional transmission of the codes is negligible, or where the language of any given message is not known and frequently changes.

4.1 Data Cleaning

Several problems arose with the UDHR samples; there were non-ascii characters, spaces, and other punctuation marks. To transform this data into a usable source, data cleaning was performed.

Unlike previously where characters wth accents or similar were simply ignored, transliteration was used to, where possible, convert non-ascii characters to their closest ascii equivalent.

To perform the cleaning, two BASH tools were used; 'sed' and 'iconv'. 'sed' is a 'stream-editor' tool, which was used to transform the data using regular expressions. 'iconv' is a text encoding conversion tool, which was used to transliterate the non-ascii characters.

The full BASH command used to clean the samples is as follows;

```
$ sed 's/[^a-zA-Z]//g' <original file> |
  iconv -f utf8 -t ascii//TRANSLIT |
  sed 's/[A-Z]/\L&/g' > <parsed file>
```

Explanation;

- 's/[^a-zA-Z]//g' : replace all characters in that aren't lower-/upper-case 'normal characters' with '' (i.e. remove them)
- iconv : We tell it to move -f (from) utf-8 encoding, -t (to) ascii encoding, using transliteration to approximate symbols where needed (e.g. é → e)
- 's/[A-Z]/\L&/g' : from the remaining lower-upper-case letters, replace the upper case with lower case equivalents

4.2 Calculating Letter Frequencies

To calculate the new letter frequencies; the cleaned sample was read in, dictionaries for letter counts for each language's sample were initialised to 0, and then the characters from the cleaned sample were looped over - updating the corresponding letter count with each loop. Once the dictionaries were populated, each letter count was divided by the length of the sample to obtain the frequencies. The dictionaries of frequencies were then converted to DataFrame objects, and the different languages DataFrames were merged into a single DataFrame in order to utilise the pre-defined functions for working with the Wikipedia data. This set of actions were performed by 'generate_letter_frequencies_ad_hoc()' from 'Project2Tools.py'.

From this new data, new binary entropies for each language were calculated, as shown below in Figure 6;

Figure 6: Table of Binary Entropies Based on Universal Declaration of Human Rights Translations

Language	Binary Entropy	Rel. Diff. from Wiki Data (%)
English	4.108452	-1.638944
French	3.900864	-3.655224
Italian	3.918025	-1.624318
German	4.023582	-1.496415

Figure 6 shows a slight decrease in binary entropy when using UDHR data, with a slightly larger decrease in French. Previously, the Huffman Codes AWLs were all within 1% of the binary entropy; if that stays true, we can expect slight reduction in the AWL for the Huffman encoded UDHR when we use the ad-hoc data.

4.3 Ad-Hoc Code Lengths for Translations of Large Text Sample

- English → Original Length: 8673, Encoded Length: 35955, Ratio: 4.145624
- French → Original Length: 9626, Encoded Length: 38046, Ratio: 3.952421
- Italian → Original Length: 10459, Encoded Length: 41253, Ratio: 3.944259
- German → Original Length: 10102, Encoded Length: 41034, Ratio: 4.061968

In each case here, we see slight improvements in the effective average word lengths, of between 0.25% and 1%. It should also be noted, that while there are improvements in AWL, the AWLs are still greater than the new binary entropies, obeying **INSERT**s theorem.

It should be noted that as the sample used to generate the letter frequencies was encoded, the AWLs estimated from the codes exactly match the effective AWLs (comparing the length of encoded data to length of original data).

5 Extensions

5.1 d-ary Huffman Codes

The efficiency of encoding data with d-ary Huffman Codes was investigated, using 'recursive_huffman()' from the 'Project2Tools.py' file, and making use of the optional 'd' parameter.

Few changes to the initial code were needed to implement this feature, they are described below;

- Define 'usable_symbols', a list of symbols that can be used in the codes; I chose to use the digits $0-9$ and the upper-case ascii characters, allowing a max(d) of 36
- Pad the initial leaf_nodes with underscores with corresponding probabilities of 0 to ensure that there are exactly d remaining items in leaf_nodes when the base case is reached
- Change from combining and removing the two least probable nodes to the d least probable nodes
- Change from assigning the codes of the least probable pair to be code_so_far + '1' or '0' to assigning the codes of the d least probable nodes to being code_so_far + one of the first d usable_symbols

Appendix Figures 10-12 contain tables of ternary, octal, and hex huffman codes, and their AWLs are shown in Figure 7;

Figure 7: Table of Average Word Lengths for Various d-ary Huffman Codes

Language	AWL Ternary	AWL Octal	AWL Hex
English	2.644760	1.464545	1.095238
French	2.529815	1.416061	1.045502
Italian	2.521178	1.407305	1.057558
German	2.605623	1.454069	1.080776

Figure 7 clearly shows that substantial improvements can be made to compression ratios if more symbols are available to encode letters, and also that as the number of symbols available grows, the AWLs tend towards 1; this is expected as there is no way to represent a letter using Huffman Coding without assigning it at least one symbol.

It should be noted from Figure 7 that there seems to be a diminishing rate of return in terms of reducing AWL for increasing d; moving from binary to ternary the AWLs reduce by roughly 1.5, while moving from ternary to octal reduces the AWLs by only just over 1. Similarly, doubling the number of symbols when moving from octal to hex only reduces the AWLs by just over 0.3. Due to this, if there is an computational expense associated with using additional symbols, a trade-off point should be calculated to choose the optimal number of symbols to minimise overall computation

While this shows a large improvement over binary Huffman Codes, d-ary storage is not always an option; binary huffman codes are the norm as classical computers can only store information in a binary format.

5.2 Letter Frequency by Usage Context

Previously it was assumed that the only impact on letter frequency was the language used. In this extension, the impact of usage context is investigated. To do this, previous results from the UDHR data were used, and counted as samples for verbose text where clarity is the most important consideration.

A second set of samples to compare these data to were then needed, where there is a restriction on sample length and so concision is the most important consideration. This set of samples was obtained by web-scraping[4]

tweets (280 character limit) tagging top newspapers in UK, France, Italy, and Germany. The code used for the web-scraping is available in file 'get_tweets.py', although it will not be immediately usable without first generating twitter api access tokens. A large sample of 10,000 tweets per newspaper was generated.

Figure 8 shows the letter frequencies for each language, for both usage contexts. It can immediately be seen that there are some large differences; for example, the Italian UDHR sample contained no 'j', 'k','w','x', or 'y' characters, while all of these occur, albeit infrequently, in the Twitter sample. This could be due to some non-Italian tweets occuring, foreign place-names being used, or product names containing these letters.

Figure 8: Table of Letter Frequencies by UDHR Source (U) and by Twitter Source (T)

Letter	English_U	English_T	French_U	French_T	German_U	German_T	Italian_U	Italian_T
a	0.081287	0.063032	0.079784	0.086877	0.058305	0.072004	0.101922	0.111778
b	0.012568	0.063577	0.007895	0.016117	0.012968	0.032845	0.008605	0.029889
c	0.033668	0.044165	0.035321	0.036151	0.031974	0.037851	0.030691	0.049392
d	0.037357	0.031191	0.045710	0.040636	0.057018	0.052269	0.050961	0.027644
e	0.124178	0.115928	0.170995	0.154602	0.174421	0.118994	0.118271	0.092663
f	0.025827	0.018663	0.008207	0.019023	0.020986	0.015256	0.008031	0.010200
g	0.019025	0.008031	0.008519	0.009212	0.037616	0.026467	0.016636	0.020219
h	0.051424	0.037142	0.004779	0.012834	0.049891	0.043100	0.008031	0.015921
i	0.080595	0.059203	0.082069	0.058703	0.076124	0.078283	0.137489	0.107714
j	0.001730	0.002759	0.001143	0.006621	0.003564	0.006611	0.000000	0.002445
k	0.002075	0.005451	0.000000	0.002541	0.012473	0.014069	0.000000	0.003130
l	0.045890	0.029003	0.057137	0.060394	0.041180	0.053444	0.064251	0.060848
m	0.021792	0.017969	0.022543	0.034284	0.016531	0.024152	0.017306	0.023354
n	0.082324	0.045549	0.079264	0.075195	0.106118	0.072303	0.074864	0.061463
o	0.081402	0.077519	0.064513	0.060711	0.020788	0.040501	0.089875	0.080407
p	0.017526	0.028386	0.025348	0.026620	0.003960	0.018992	0.023616	0.038562
q	0.001845	0.000898	0.008311	0.006449	0.000000	0.002049	0.003155	0.003767
r	0.070218	0.084724	0.065032	0.073923	0.078697	0.071540	0.063773	0.068565
s	0.053615	0.063573	0.075317	0.073131	0.055039	0.069193	0.048857	0.048299
t	0.092586	0.137975	0.089549	0.070335	0.060087	0.074788	0.070274	0.075114
u	0.022599	0.016747	0.052566	0.041327	0.050980	0.031645	0.033655	0.030906
v	0.011415	0.016240	0.009350	0.015484	0.008513	0.009883	0.013386	0.016406
w	0.008648	0.014841	0.000000	0.002060	0.010790	0.014649	0.000000	0.004223
x	0.000807	0.001540	0.004571	0.005981	0.000099	0.002244	0.000000	0.002130
y	0.019140	0.014598	0.002078	0.007950	0.000198	0.006645	0.000000	0.003052
z	0.000461	0.001297	0.000000	0.002836	0.011681	0.010220	0.016350	0.011910

The difference between the two samples in terms of spread of letter frequencies was investigated for each language, in addition to calculating their absolute and relative differences. The results are shown below;

Figure 9: Table Comparing Letter Frequency Statistics Between UDHR and Twitter Samples

Language	s.t.d. probs UDHR	s.t.d. probs Twitter	mean abs diffs in probs	mean rel diffs in probs
English	0.034748	0.036080	0.012247	0.435794
French	0.041554	0.036357	0.006708	0.356082
Italian	0.040242	0.034156	0.008671	0.402040
German	0.039772	0.030283	0.011201	0.382233

Figure 9 shows sightly lower standard deviation in letter frequencies for the Twitter samples than for the UDHR samples. This could be because tweets contain information about a wider range of topics than the UDHR, leading to more opportunities / situations where 'uncommon' letters may be used, especially in product names.

Additionally, it can be seen from Figure 9 that for each language there is on average a $35-45\%$ difference in probability for each letter between the two samples; this is almost as much of a difference as between languages (around 60%). Due to this, an especially efficient way to encode messages would be to generate letter frequencies by using samples of text in the same language, and in the same context, as the messages.

6 Appendix

6.1 d-ary Code Tables

Figure 10: Table of Ternary Huffman Codes

Letter	English_codes	French_codes	Italian_codes	German_codes
a	01	02	11	212
b	1221	20112	2100	2000
c	120	112	021	012
d	121	200	212	211
e	20	21	12	22
f	2222	1100	22021	011
g	2201	1102	2200	121
h	211	20110	22022	201
i	00	10	20	02
j	122211	2011102	2202000	200102
k	12220	20111010	2202001	1202
l	210	220	222	122
m	2220	2012	2201	2002
n	10	01	01	11
o	02	221	10	010
p	2200	111	020	20011
q	122212	1101	220202	2001000
r	221	222	221	10
s	212	00	211	210
t	11	12	00	00
u	2221	202	022	202
v	1220	2010	2101	20012
w	12222	20111011	2202002	1200
x	1222102	201112	2202010	2001002
y	2202	201111	2202011	200101
z	1222101	20111012	2102	1201

Figure 11: Table of Octal Huffman Codes

Letter	English_codes	French_codes	Italian_codes	German_codes
a	0	0	0	0
b	63	753	64	65
c	72	72	72	71
d	73	73	75	77
e	1	1	1	1
f	71	754	62	70
g	65	756	67	72
h	75	752	63	74
i	2	2	2	2
j	605	7506	600	606
k	607	7500	601	64
l	74	76	77	73
m	67	70	70	66
n	3	3	3	3
o	4	77	4	67
p	64	71	71	607
q	606	755	61	600
r	77	4	76	4
s	76	5	74	76
t	5	6	5	5
u	70	74	73	75
v	62	757	65	61
w	61	7501	602	62
x	604	751	603	604
y	66	7507	604	605
z	603	7502	66	63

Figure 12: Table of Hex Huffman Codes

Letter	English_codes	French_codes	Italian_codes	German_codes
a	0	0	0	0
b	FC	FC	FD	FE
c	1	1	1	1
d	2	2	2	2
e	3	3	3	3
f	4	FD	FB	4
g	FE	FF	4	5
h	5	FB	FC	6
i	6	4	5	7
j	F7	F8	F0	F8
k	F9	F0	F1	FD
l	7	5	6	8
m	8	6	7	FF
n	9	7	8	9
o	A	8	9	A
p	FD	9	A	F9
q	F8	FE	FA	F0
r	B	A	B	B
s	C	B	C	C
t	D	C	D	D
u	E	D	E	E
v	FB	E	FE	FA
w	FA	F1	F2	FB
x	F6	FA	F3	F6
y	FF	F9	F4	F7
z	F5	F2	FF	FC

References

[1] Khalid Sayood.
Introduction to Data Compression, Third Edition.
Editor: Edward A. Fox, Virginia Polytechnic University, 2006.

[2] David MacKay.
Information Theory, Inference, and Learning Algorithms.
Cambridge University Press, 2003.

[3] Matt Mahoney.
Rationale for a Large Text Compression Benchmark.
`https://cs.fit.edu/ mmahoney/compression/rationale.html`
Access Date: 30/12/2018

[4] Janet Williams.
How to Scrape Data from Twitter Using Python
`https://www.promptcloud.com/blog/scrape-twitter-data-using-python-r`
Access Date: 30/12/2018