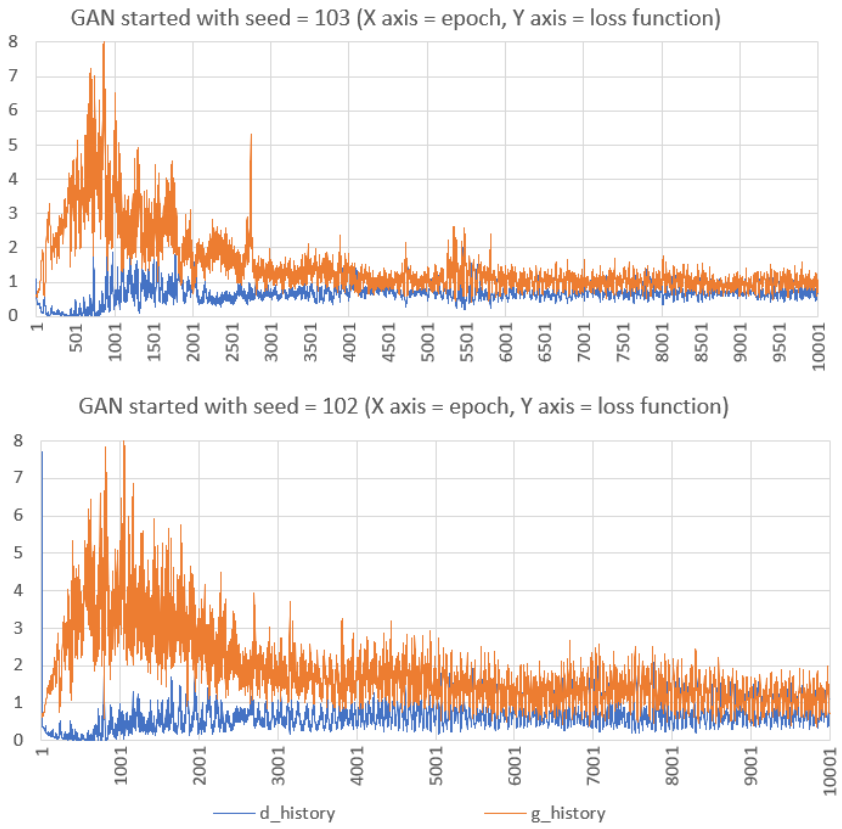
Practical AI & Machine Learning Projects and Datasets



Vincent Granville, Ph.D. | www.MLTechniques.com | Version 3.0, December 2023

Preface

This book is intended to participants in the AI and machine learning certification program organized by my AI/ML research lab MLtechniques.com. It is also an invaluable resource to instructors and professors teaching related material, and to their students. If you want to add enterprise-grade projects to your curriculum, with deep technical dive on modern topics, you are welcome to re-use my projects in your classroom. I provide my own solution to each of them.

This book is also useful to prepare for hiring interviews. And for hiring managers, there is plenty of original questions, encouraging candidates to think outside the box, with applications on real data. The amount of Python code accompanying the solutions is considerable, using a vast array of libraries as well as home-made implementations showing the inner workings and improving existing black-box algorithms. By itself, this book constitutes a solid introduction to Python and scientific programming. The code is also on my GitHub repository.

The topics cover generative AI, synthetic data, machine learning optimization, scientific computing with Python, experimental math, synthetic data and functions, data visualizations and animations, time series and spatial processes, NLP and large language models, as well as graph applications and more. It also includes significant advances on some of the most challenging mathematical conjectures, obtained thanks to modern computations. In particular, intriguing new results regarding the Generalized Riemann Hypothesis, and a conjecture regarding record run lengths in the binary digits of $\sqrt{2}$. For the latter, the author offers a \$1m award to prove or disprove the main statement. Most projects are based on real life data, offered with solutions and Python code. Your own solutions would be a great addition to your GitHub portfolio, bringing your career to the next level. Hiring managers, professors, and instructors can use the projects, each one broken down in a number of steps, to differentiate themselves from competitors. Most offer off-the-beaten path material. They may be used as novel exercises, job interview or exam questions, and even research topics for master or PhD theses.

To see how the certification program works, check out our FAQ posted here, or click on the "certification" tab on our website MLtechniques.com. Certifications can easily be displayed on your LinkedIn profile page in the credentials section. Unlike many other programs, there is no exam or meaningless quizzes. Emphasis is on projects with real-life data, enterprise-grade code, efficient methods, and modern applications to build a strong portfolio and grow your career in little time. The guidance to succeed is provided by the founder of the company, one of the top experts in the field, Dr. Vincent Granville. Jargon and unnecessary math are avoided, and simplicity is favored whenever possible. Nevertheless, the material is described as advanced by everyone who looked at it.

The related teaching and technical material (textbooks) can be purchased at MLtechniques.com/shop/. MLtechniques.com, the company offering the certifications, is a private, self-funded AI/ML research lab developing state-of-the-art open source technologies related to synthetic data, generative AI, cybersecurity, geospatial modeling, stochastic processes, chaos modeling, and AI-related statistical optimization.

About the author

Vincent Granville is a pioneering data scientist and machine learning expert, co-founder of Data Science Central (acquired by TechTarget), founder of MLTechniques.com, former VC-funded executive, author and patent owner.

Vincent's past corporate experience includes Visa, Wells Fargo, eBay, NBC, Microsoft, and CNET. Vincent is also a former post-doc at Cambridge University, and the National Institute of Statistical Sciences (NISS). He published in *Journal of Number Theory*, *Journal of the Royal Statistical Society* (Series B), and *IEEE Transactions on Pattern Analysis and Machine Intelligence*. He is also the author of multiple books, available here. He lives in Washington state, and enjoys doing research on stochastic processes, dynamical systems, experimental math and probabilistic number theory.

Contents

Chapter 7

NLP and Large Language Models

If you tried apps such as GPT (generative pre-training transformer), you may be surprised by the quality of the sentences, images, sound, videos, or code generated. Yet, in the end, the value is the depth and relevance or the content generated, more than the way it is presented. My interest started when I did a Google search for "variance of the range for Gaussian distributions". I vaguely remember that it is of the order $1/\sqrt{n}$ where n is the number of observations, but could not find the reference anymore. Indeed I could not find anything at all on this topic. The resources I found on the subject 10 years ago are all but gone, or at least very hard to find. As search evolved over time, it now caters to a much larger but less educated audience. As a result, none of the search results were even remotely relevant to my question. This is true for pretty much any research question that I ask.

Using OpenAI, I found the answer I was looking for, even with more details than expected, yet with no link to an actual reference, no matter how I change my prompt. OpenAI could not find my answer right away, and I had to rephrase my prompt as "what is the asymptotic variance of the range for Gaussian distributions". More general prompts on specific websites, such as "asymptotic distribution of sample variance" lead to a number of articles which in turn lead to some focusing on Gaussian distributions. Even today, automatically getting a good answer in little time, with a link, is still a challenge.

In this chapter, one project focuses on this issue. Smart, optimized crawling is part of the solution, combined with using OpenAI output or trying to reverse-engineer OpenAI to identify input sources. But the goal is not to create a new version of OpenAI. Rather, do what it can't do, or what it refuses to do for legal reasons. The other projects involve making predictions or synthetizations based on unstructured data repositories, mostly consisting of text, while scoring the input sources and the output. This is a less well-known aspect of large language models (LLM), with a focus on structuring unstructured data, scoring content, and creating taxonomies.

7.1 Synthesizing DNA sequences with LLM techniques

This project is not focused on genome data alone. The purpose is to design a generic solution that may also work in other contexts, such as synthesizing molecules. The problem involves dealing with a large amount of "text". Indeed the sequences discussed here consists of letter arrangements, from an alphabet that has 5 symbols: A, C, G, T and N. The first four symbols stand for the types of bases found in a DNA molecule: adenine (A), cytosine (C), guanine (G), and thymine (T). The last one (N) represents missing data. No prior knowledge of genome sequencing is required.

The data consists of DNA sub-sequences from an number of individuals, and categorized according to the type of genetic patterns found in each sequence. Here I combined the sequences together. The goal is to synthesize realistic DNA sequences, evaluate the quality of the synthetizations, and compare the results with random sequences. The idea is to look at a DNA string S_1 consisting of n_1 consecutive symbols, to identify potential candidates for the next string S_2 consisting of n_2 symbols. Then, assign a probability to each string S_2 conditionally on S_1, use these transition probabilities to sample S_2 given S_1, then move to the right by n_2 symbols, do it again, and so on. Eventually you build a synthetic sequence of arbitrary length. There is some analogy with Markov chains. Here, n_1 and n_2 are fixed, but arbitrary.

7.1.1 Project and solution

Let's look at 3 different DNA sequences. The first one is from a real human being. The second one is synthetic, replicating some of the patterns found in real data. The third one is purely random, with each letter indepen-

```
Real DNA

ATGCCCCAACTAAATACTACCGTATGGCCCACCATAATTACCCCCATACTCCTTACACTATTCCTCATCACCCAACTA
AAAATATTAAACACAAACTACCACCTACCTCCCTCACCAAAGCCCATAAAAATAAAAAATTATAACAAACCCTGAGAA
CCAAAATGAACGAAAATCTGTTCGCTTCATTCATTGCCCCCACAATCCTAGNATGAACGAAAATCTGTTCGCTTCATT
CATTGCCCCCACAATCCTAGGCCTACCCGCCGCAGTACTGATCATTCTATTTCCCCCTCTATTGATCCCCACCTCCAA
ATATCTCATCAACAACCGACTAATCACCACCCAACAATGACTAATCAAACTAACCTCAAAACAAATGATAACCATACA

Synthetic DNA

TTGTTTTCTTCACCTAAATGCACAAGAATGGTGGGCCGAGGAGCCATGTCAAGTGGGGATGGGTCTATCGAACCTGAG
GGCCCCCCACTTCAGATGCTTCGTACTGTCTTTGGGACTTCTCACCGTCTCATGGTCTGCCCTGCCCCGCAGTGTGGC
CTGGTATTTTTAACCCTATTATAGAAACAACAATTTATGGGCTCCTTGAAGCTTATACAATACAACAGTAAAGGGCCC
CTCCTCCAGTCAGCCTCTTTCCCTCTTAGGGTAAATGAGGATATCCAAGTGCCCACCTCATCATCAACTCCGCCACCA
GTTTGCAGCCCTTGCAGGAGATTTCTGGTGATGAAAGTTCAGTGGACTTGGGAAAAGCCGTCATGCTGTCTGCCAACC

Random DNA

ATCCTGCTTCATATGTAGGAAGGGTTGTAGGTTCCCGGAGGGCGCATTGCAAAGACCGGCCAGACTACTTATGGCCGC
GTCCTAAGCACCATATGCTAAGCCTGATTAACATCGCGCGGATGTAACTACACGCGCGCTACGTGAATCCTAGGCAGC
CGTCACGATTGACTCCTCATACTCATCGAGGCGCTCGCGTCATAGACCGACCATCGCGTCACCATAATAAGTAGAGTC
TTTACGGTAGGCCTTCAAAATACGGACAAGGCATTTGTATTCTTCATGTCATGTAGCTGAAGAATACCATTAAGTTTA
TAGGCGGGTGTACGACAAGACTGCCAGGTGGCAGTGTCGTCACAAGAGCGCGTAAACTTTTTGCCGGTAATAGACCGT
```

Table 7.1: Genome data, three DNA sub-sequences: real, synthetic, and random

dently distributed from each other, with the same 25% marginal frequency. Can you tell the differences just by looking at the 3 sequences in Table 7.1? If not, see Figure 7.1 and accompanying description.

For this project, I start with real sequences to train the DNA synthesizer. I then evaluate the quality, and show how synthetic DNA is superior to random sequences. Any symbol other than A, C, G, or T must be labeled as N. It represents missing data and I ignore it in the Python code. Figures 7.1 and 7.2 illustrate the end result: comparing string frequencies in real, synthetic and random DNA sequences, for specific "words" consisting of n_3 consecutive symbols. The main diagonal (red line) represents perfect fit with real DNA. The number of "words" (dots in the pictures) varies from 4000 to 10,000, and are called `nodes` in the Python code. It shows that these DNA sequences are anything but random, with synthetic fitting real data quite well.

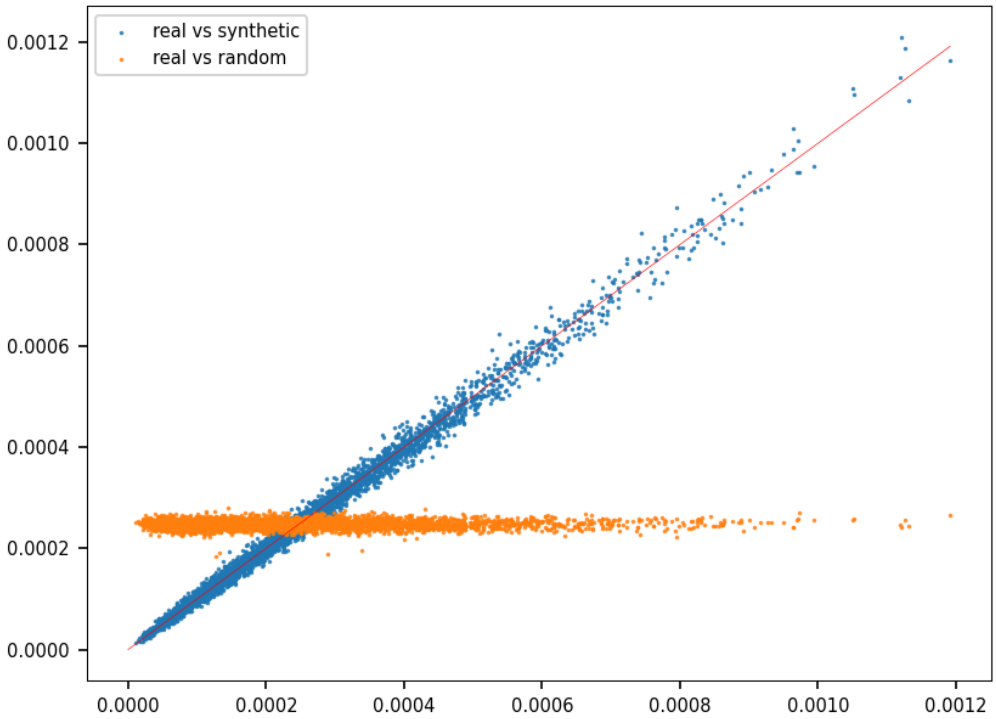


Figure 7.1: PDF scatterplots, $n_3 = 6$: real DNA vs synthetic (blue), and vs random (orange)

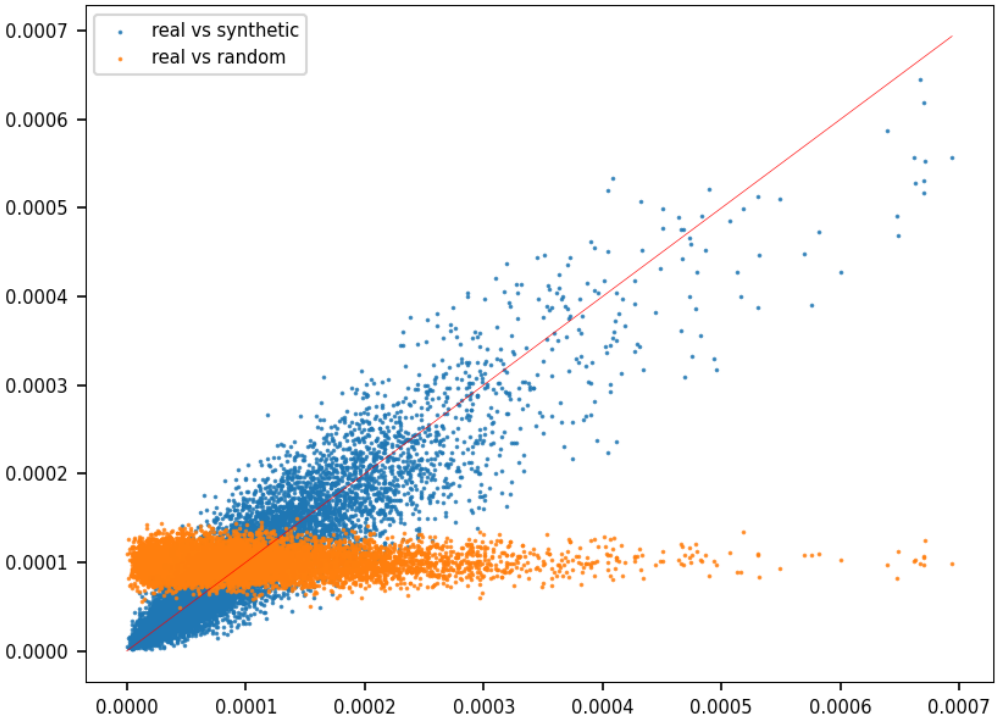


Figure 7.2: PDF scatterplots, $n_3 = 8$: real DNA vs synthetic (blue), and vs random (orange)

The project consists of the following steps:

Step 1: Understanding the data. Look at the URL in the Python code in section 7.1.2 to access the data. Ignore the "class" feature as the purpose here is not to classify DNA sequences. A small extract of a real DNA sequence is featured in Table 7.1, at the top. Note that the order of the letters is important.

Step 2: Compute summary statistics. For n_1 and n_2 fixed, extract all distinct strings S_1, S_2 of length respectively n_1 and n_2, and compute their occurrences. Do the same for strings S_{12} of length $n_1 + n_2$. The goal is to predict, given a string S_1 of length n_1, the probability to be followed by a specific string S_2 of length n_2. That is, $P[S_2 = s_2|S_1 = s_1]$. Here a string is a sub-sequence of letters. Strings are called words, and letters are also called characters. The four letters are 'A', 'B', 'C', 'D'. Ignore the letter 'N'. Then, do the same when the strings S_1 and S_2 are separated by a gap of g letters, for $g = 1, 2, 3$.

Step 3: String associations. Two specific strings S_1, S_2 may frequently be found together, or rarely. Characterize this string association using the pointwise mutual information (PMI) [Wiki]. Rare occurrences may indicate a rare genetic condition. Order the (S_1, S_2) found in the data according to the PMI metric.

Step 4: Synthesize a DNA sequence. Proceed as follows. Start with an arbitrary string S_1 of length n_1. Then add n_2 letters at a time, sequentially, to the DNA sequence being generated. In other words, at each step, sample S_2 from $P(S_2|S_1)$, where S_2 is the new string of length n_2 to be added, and S_1 is the last string of length n_1 built so far.

Step 5: Evaluate the quality. In this context, the Hellinger distance [Wiki] is simple and convenient, to assess the quality of the synthetic DNA sequence, that is, how well it replicates the patterns found in real DNA. The value is between 0 (great fit) and 1 (worst fit). Randomly select $n = 10{,}000$ strings S_3 of length n_3 found in real DNA. These strings are referred to as *nodes*. Compute the frequency $P_{\text{real}}(S_3)$ for each of them. Also compute the frequency $P_{\text{synth}}(S_3)$ on the synthetic sequence. The Hellinger distance is then

$$\text{HD} = \sqrt{1 - \sum \sqrt{P_{\text{real}}(S_3) \cdot P_{\text{synth}}(S_3)}},$$

where the sum is over all the selected strings S_3. Also compare real DNA with a random sequence, using HD. Show that synthetic DNA is a lot better than random sequences, to mimic real DNA. Finally, try different values of n_1, n_2, n_3 and check whether using $n = 1000$ nodes provides a good enough approximation to HD (it is much faster than $n = 10{,}000$, especially when n_3 is large).

The solution to the first four steps correspond to steps [1–4] in the Python code in section 7.1.2, while Step 5 corresponds to step [6] in the code. To compute summary statistics (Step 2) when S_1 and S_2 are separated by a gap of g letters, replace `string2=obs[pos1:pos2]` by `string2=obs[pos1+g:pos2+g]` in step [2] in the code. The interest in doing this is to assess whether there are long-range associations between strings. By default, in the current version of the code, $g = 0$.

Figures 7.1 and 7.2 show scatterplots with probability vectors $[P_{\text{real}}(S_3), P_{\text{synth}}(S_3)]$ in blue, for thousands of strings S_3 found in the real DNA sequence. For orange dots, the second component $P_{\text{synth}}(S_3)$ is replaced by $P_{\text{rand}}(S_3)$, the value computed on a random sequence. Clearly, the synthetic DNA is much more realistic than the random DNA, especially when $n_3 = 6$. Note that the probabilities are associated to overlapping events: for instance, the strings 'AACT' and 'GAAC' are not independent, even in the random sequence. The Hellinger distance used here is not adjusted for this artifact.

7.1.2 Python code

The code is also on GitHub, here. For explanations, see section 7.1.1.

```
# genome.py : synthesizing DNA sequences
# data: https://www.kaggle.com/code/tarunsolanki/classifying-dna-sequence-using-ml

import pandas as pd
import numpy as np
import re # for regular expressions


#--- [1] Read data

url = "https://raw.githubusercontent.com/VincentGranville/Main/main/dna_human.txt"
human = pd.read_table(url)
# human = pd.read_table('dna_human.txt')
print(human.head())


#--- [2] Build hash table architecture
#
# hash1_list[string1] is the list of potential string2 found after string1, separated by
    ~

nobs = len(human)
print(nobs)
hash12 = {}
hash1_list = {}
hash1 = {}
hash2 = {}
count1 = 0
count2 = 0
count12 = 0
sequence = ''

for k in range(nobs):
  obs = human['sequence'][k]
  sequence += obs
  sequence += 'N'
  type = human['class'][k]
  length = len(obs)
  string1_length = 4
  string2_length = 2
  pos0 = 0
  pos1 = pos0 + string1_length
  pos2 = pos1 + string2_length

  while pos2 < length:

     string1 = obs[pos0:pos1]
     string2 = obs[pos1:pos2]
```

```
      if string1 in hash1:
         if string2 not in hash1_list[string1] and 'N' not in string2:
            hash1_list[string1] = hash1_list[string1] + '~' + string2
         hash1[string1] += 1
         count1 += 1
      elif 'N' not in string1:
         hash1_list[string1] = '~' + string2
         hash1[string1] = 1
      key = (string1, string2)

      if string2 in hash2:
         hash2[string2] += 1
         count2 += 1
      elif 'N' not in string2:
         hash2[string2] = 1

      if key in hash12:
         hash12[key] += 1
         count12 += 1
      elif 'N' not in string1 and 'N' not in string2:
         hash12[key] = 1

      pos0 += 1
      pos1 += 1
      pos2 += 1

   if k % 100 == 0:
      print("Creating hash tables: %6d %6d %4d" %(k, length, type))


#--- [3] Create table of string associations, compute PMI metric

print()
index = 0
for key in hash12:
   index +=1
   string1 = key[0]
   string2 = key[1]
   n1 = hash1[string1] # occurrences of string1
   n2 = hash2[string2] # occurrences of string2
   n12 = hash12[key] # occurrences of (string1, string2)
   p1 = n1 / count1 # frequency of string1
   p2 = n2 / count2 # frequency of string2
   p12 = n12 / count12 # frequency of (string1, string2)
   pmi = p12 / (p1 * p2)
   if index % 100 == 0:
      print("Computing string frequencies: %5d %4s %2s %4d %8.5f"
            %(index, string1, string2, n12, pmi))
print()


#--- [4] Synthetization
#
# synthesizing word2, one at a time, sequencially based on previous word1

n_synthetic_string2 = 2000000
seed = 65
np.random.seed(seed)

synthetic_sequence = 'TTGT' # starting point (must be existing string1)
pos1 = len(synthetic_sequence)
pos0 = pos1 - string1_length


for k in range(n_synthetic_string2):
```

```
    string1 = synthetic_sequence[pos0:pos1]
    string = hash1_list[string1]
    myList = re.split('~', string)

    # get target string2 list in arr_string2, and corresponding probabilities in arr_proba
    arr_string2 = []
    arr_proba = []
    cnt = 0
    for j in range(len(myList)):
        string2 = myList[j]
        if string2 in hash2:
            key = (string1, string2)
            cnt += hash12[key]
            arr_string2.append(string2)
            arr_proba.append(hash12[key])
    arr_proba = np.array(arr_proba)/cnt

    # build cdf and sample word2 from cdf, based on string1
    u = np.random.uniform(0, 1)
    cdf = arr_proba[0]
    j = 0
    while u > cdf:
        j += 1
        cdf += arr_proba[j]
    synthetic_string2 = arr_string2[j]
    synthetic_sequence += synthetic_string2
    if k % 100000 == 0:
        print("Synthesizing %7d/%7d: %4d %8.5f %2s"
              % (k, n_synthetic_string2, j, u, synthetic_string2))

    pos0 += string2_length
    pos1 += string2_length

print()
print("Real DNA:\n", sequence[0:1000])
print()
print("Synthetic DNA:\n", synthetic_sequence[0:1000])
print()


#--- [5] Create random sequence for comparison purposes

print("Creating random sequence...")
length = (1 + n_synthetic_string2) * string2_length
random_sequence = ""
map = ['A', 'C', 'T', 'G']

for k in range(length):
    random_sequence += map[np.random.randint(4)]
    if k % 100000 == 0:
        print("Creating random sequence: %7d/%7d" %(k,length))
print()
print("Random DNA:\n", random_sequence[0:1000])
print()


#--- [6] Evaluate quality: real vs synthetic vs random DNA

max_nodes = 10000 # sample strings for frequency comparison
string_length = 6 # length of sample strings (fixed length here)

nodes = 0
hnodes = {}
iter = 0
```

```
while nodes < max_nodes and iter < 5*max_nodes:
   index = np.random.randint(0, len(sequence)-string_length)
   string = sequence[index:index+string_length]
   iter += 1
   if string not in hnodes and 'N' not in string:
      hnodes[string] = True
      nodes += 1
      if nodes % 1000 == 0:
         print("Building nodes: %6d/%6d" %(nodes, max_nodes))
print()

def compute_HD(hnodes, sequence, synthetic_sequence):

   pdf1 = []
   pdf2 = []
   cc = 0

   for string in hnodes:
      cnt1 = sequence.count(string)
      cnt2 = synthetic_sequence.count(string)
      pdf1.append(float(cnt1))
      pdf2.append(float(cnt2))
      ratio = cnt2 / cnt1
      if cc % 100 == 0:
         print("Evaluation: computing EPDFs: %6d/%6d: %5s %8d %8d %10.7f"
               %(cc, nodes, string, cnt1, cnt2, ratio))
      cc += 1

   pdf1 = np.array(pdf1) # original dna sequence
   pdf2 = np.array(pdf2) # synthetic dna sequence
   pdf1 /= np.sum(pdf1)
   pdf2 /= np.sum(pdf2)

   HD = np.sum(np.sqrt(pdf1*pdf2))
   HD = np.sqrt(1 - HD)
   return(pdf1, pdf2, HD)

pdf_dna, pdf_synth, HD_synth = compute_HD(hnodes, sequence, synthetic_sequence)
pdf_dna, pdf_random, HD_random = compute_HD(hnodes, sequence, random_sequence)

print()
print("Total nodes: %6d" %(nodes))
print("Hellinger distance [synthetic]: HD = %8.5f" %(HD_synth))
print("Hellinger distance [random] : HD = %8.5f" %(HD_random))

#--- [7] Visualization (PDF scatterplot)

import matplotlib.pyplot as plt
import matplotlib as mpl

mpl.rcParams['axes.linewidth'] = 0.5
plt.rcParams['xtick.labelsize'] = 7
plt.rcParams['ytick.labelsize'] = 7

plt.scatter(pdf_dna, pdf_synth, s = 0.1, color = 'red', alpha = 0.5)
plt.scatter(pdf_dna, pdf_random, s = 0.1, color = 'blue', alpha = 0.5)
plt.legend(['real vs synthetic', 'real vs random'], loc='upper left', prop={'size': 7}, )
plt.plot([0, np.max(pdf_dna)], [0, np.max(pdf_dna)], c='black', linewidth = 0.3)
plt.show()
```

Bibliography

[1] Adel Alamadhi, Michel Planat, and Patrick Solé. Chebyshev's bias and generalized Riemann hypothesis. *Preprint*, pages 1–9, 2011. arXiv:1112.2398 [Link]. 54

[2] K. Binswanger and P. Embrechts. Longest runs in coin tossing. *Insurance: Mathematics and Economics*, 15:139–149, 1994. [Link]. 47

[3] Ramiro Camino, Christian Hammerschmidt, and Radu State. Generating multi-categorical samples with generative adversarial networks. *Preprint*, pages 1–7, 2018. arXiv:1807.01202 [Link]. 83

[4] Fida Dankar et al. A multi-dimensional evaluation of synthetic data generators. *IEEE Access*, pages 11147–11158, 2022. [Link]. 82

[5] Antónia Földes. The limit distribution of the length of the longest head-run. *Periodica Mathematica Hungarica*, 10:301–310, 1979. [Link]. 48

[6] Louis Gordon, Mark F. Schilling, and Michael S. Waterman. An extreme value theory for long head runs. *Probability Theory and Related Fields*, 72:279–287, 1986. [Link]. 47

[7] Vincent Granville. Feature clustering: A simple solution to many machine learning problems. *Preprint*, pages 1–6, 2023. MLTechniques.com [Link]. 71

[8] Vincent Granville. Generative AI: Synthetic data vendor comparison and benchmarking best practices. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 64

[9] Vincent Granville. Generative AI technology break-through: Spectacular performance of new synthesizer. *Preprint*, pages 1–16, 2023. MLTechniques.com [Link]. 12, 15, 95

[10] Vincent Granville. *Gentle Introduction To Chaotic Dynamical Systems*. MLTechniques.com, 2023. [Link]. 49, 53

[11] Vincent Granville. How to fix a failing generative adversarial network. *Preprint*, pages 1–10, 2023. MLTechniques.com [Link]. 14

[12] Vincent Granville. Massively speed-up your learning algorithm, with stochastic thinning. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 13, 71

[13] Vincent Granville. Smart grid search for faster hyperparameter tuning. *Preprint*, pages 1–8, 2023. MLTechniques.com [Link]. 13, 69, 71

[14] Vincent Granville. *Synthetic Data and Generative AI*. Elsevier, 2024. [Link]. 27, 32, 33, 34, 36, 37, 43, 44, 46, 52, 63, 64, 68, 69, 71, 72, 74, 82, 83

[15] Elisabeth Griesbauer. *Vine Copula Based Synthetic Data Generation for Classification*. 2022. Master Thesis, Technical University of Munich [Link]. 71

[16] Emil Grosswald. Oscillation theorems of arithmetical functions. *Transactions of the American Mathematical Society*, 126:1–28, 1967. [Link]. 54

[17] Adam J. Harper. Moments of random multiplicative functions, II: High moments. *Algebra and Number Theory*, 13(10):2277–2321, 2019. [Link]. 54

[18] Adam J. Harper. Moments of random multiplicative functions, I: Low moments, better than squareroot cancellation, and critical multiplicative chaos. *Forum of Mathematics, Pi*, 8:1–95, 2020. [Link]. 54

[19] Adam J. Harper. Almost sure large fluctuations of random multiplicative functions. *Preprint*, pages 1–38, 2021. arXiv [Link]. 54

[20] Zsolt Karacsony and Jozsefne Libor. Longest runs in coin tossing. teaching recursive formulae, asymptotic theorems and computer simulations. *Teaching Mathematics and Computer Science*, 9:261–274, 2011. [Link]. 48

[21] Tamas Mori. The a.s. limit distribution of the longest head run. *Canadian Journal of Mathematics*, 45:1245–1262, 1993. [Link]. 48

[22] Michel Planat and Patrick Solé. Efficient prime counting and the Chebyshev primes. *Preprint*, pages 1–15, 2011. arXiv:1109.6489 [Link]. 54

[23] M.S. Schmookler and K.J. Nowka. Bounds on runs of zeros and ones for algebraic functions. *Proceedings 15th IEEE Symposium on Computer Arithmetic*, pages 7–12, 2001. ARITH-15 [Link]. 47

[24] Mark Shilling. The longest run of heads. *The College Mathematics Journal*, 21:196–207, 2018. [Link]. 47

[25] Chang Su, Linglin Wei, and Xianzhong Xie. Churn prediction in telecommunications industry based on conditional Wasserstein GAN. *IEEE International Conference on High Performance Computing, Data, and Analytics*, pages 186–191, 2022. IEEE HiPC 2022 [Link]. 82

[26] Terence Tao. Biases between consecutive primes. *Tao's blog*, 2016. [Link]. 54

[27] Ruonan Yu, Songhua Liu, and Xinchao Wang. Dataset distillation: A comprehensive review. *Preprint*, pages 1–23, 2022. Submitted to IEEE PAMI [Link]. 69

Index