**How to distinguish between structured and random signals in Python**

[[Anatoly Alekseev](https://medium.com/@fingoldo?source=post_page-----15819b261de0--------------------------------)](https://medium.com/@fingoldo?source=post_page-----15819b261de0--------------------------------)

[Anatoly Alekseev](https://medium.com/@fingoldo?source=post_page-----15819b261de0--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F33768f06b3b4&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40fingoldo%2Fhow-to-distinguish-between-structured-and-random-signals-in-python-15819b261de0&user=Anatoly+Alekseev&userId=33768f06b3b4&source=post_page-33768f06b3b4----15819b261de0---------------------post_header-----------)

5 min read

·

Apr 14

1

Distinguishing random from structured signals is a fundamental task in statistics, machine learning, and data science in general, as it enables us to understand the underlying patterns and relationships in our data.



Image generated by [Kandinsky 2.1](https://rudalle.ru/kandinsky2) when asked to visualize random and structured signals

Random signals are characterized by unpredictable and chaotic behavior, while structured signals exhibit regular patterns and dependencies. By telling these two types of signals apart, we can gain insights into the underlying processes that generated the data, and make more informed decisions.

In machine learning, difference between random and structured signals is crucial for building predictive models that can generalize well to new data. If a model is built solely on random noise, it will not be able to accurately predict on new data. On the other hand, if the model captures the underlying structure of the data, it can indeed be used to make accurate predictions and uncover relationships between variables.

In this article, I would like to demonstrate the usage of various kinds of statistical measures called **entropies**on a synthetic dataset where we know exact origin of our data, and advise an aspiring data scientist two existing Python packages that offer a wide coverage of entropy flavors.

[Entropy](https://en.wikipedia.org/wiki/Entropy) is a concept used in statistics, information theory, and physics to quantify the amount of disorder/randomness in a system. There are several kinds of entropy that are computed differently, but bear similar meaning. For a more detailed review of them, I recommend “[Measuring Signal Complexity/Regularity](https://www.youtube.com/watch?v=5vOYgJ-80Bg)” video by Rami Khushaba.

For the purpose of this article, we will create a structured (i.e., produced by a strict functional mapping), a noisy (i.e., plus som small random noise), a random (i.e., produced by the pseudo random generator) and a “semi-structured” (in our case, it will be the structured, but randomly shuffled) data series, and check if we’ll be able to distinguish between them using aforementioned information-theoretic quantities.

def alter\_series(series: dict, name: str):  
 "Adds noisy and shuffled options of the series"  
 series["noisy\_" + name] = series[name] + (np.random.random(N) - 0.5) / 4  
 series["shuffled\_" + name] = series[name].copy()  
 np.random.shuffle(series["shuffled\_" + name])  
  
  
def create\_samples() -> tuple:  
 "Create a structured, a noisy, a random, and a structured but randomly shuffled 1D arrays"  
  
 series = {}  
 series["constant"] = np.ones(N)  
 series["line"] = np.arange(N) \* 0.01 - 0.4  
 alter\_series(series, "line")  
  
 series["parabola"] = np.arange(N) \*\* 2 / 200  
 alter\_series(series, "parabola")  
  
 series["sine"] = np.sin(np.arange(N))  
 alter\_series(series, "sine")  
  
 series["random\_uniform"] = np.random.uniform(size=N)  
 series["random\_normal"] = np.random.normal(size=N)  
 series["random\_lognormal"] = np.random.lognormal(size=N)  
  
 plt.figure(figsize=(10, 5))  
 for var\_name, var\_data in series.items():  
 if "random" in var\_name:  
 linestyle = "dotted"  
 alpha = 0.6  
 elif "shuffled" in var\_name:  
 continue  
 linestyle = "dashdot"  
 alpha = 0.3  
 elif "nois" in var\_name:  
 linestyle = "dashed"  
 alpha = 1  
 else:  
 alpha = 1  
 linestyle = "solid"  
 plt.plot(var\_data, alpha=alpha, label=var\_name, linestyle=linestyle)  
  
 plt.xlim(0, N)  
 plt.ylim(-2, 2)  
 plt.legend()  
  
 return series

**Generating structured, noisy, random random, semi-structured short series**

N = 60  
series = create\_samples()

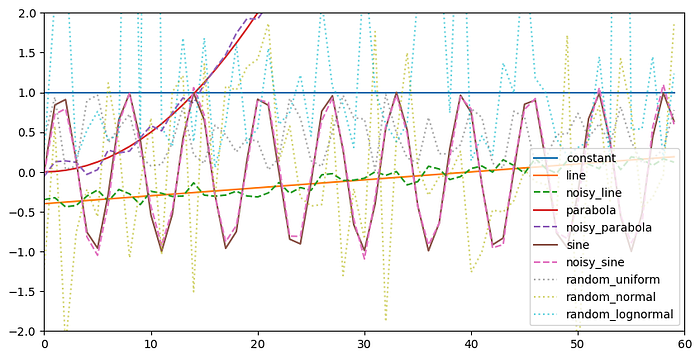


Figure 1. Our structured, noisy, random, and shuffled series of short size. Produced by the author.

**Comparing different entropies on all 3 short series**

We will be using one custom written function computing vanilla Shannon’s entropy:

def naive\_entropy(x):  
 “Naive Shannon entropy implementation”  
 vals, counts = np.unique(  
 x,  
 return\_counts=True,  
 )  
return entropy(counts)

But our main toolbox will be provided by already existing packages [antropy](https://github.com/raphaelvallat/antropy" \t "_blank) and [EntropyHub](https://www.entropyhub.xyz/" \t "_blank), that (especially the latter) cover more than a mere mortal can desire.

At first I only wanted to compare entropies mentioned by Rami in his video, but what the heck, let’s try all what our packages have to offer:

antropy\_functions = [fn for fn in dir(ant) if fn.endswith("\_entropy")]  
print(antropy\_functions)  
#['app\_entropy', 'perm\_entropy', 'sample\_entropy', 'spectral\_entropy', 'svd\_entropy']  
  
EntropyHub\_functions = [fn for fn in dir(EH) if fn.endswith("En") and "\_" not in fn]  
print(EntropyHub\_functions)  
#['ApEn', 'AttnEn', 'BubbEn', 'CoSiEn', 'CondEn', 'DispEn', 'DistEn', 'EnofEn', 'FuzzEn', 'GridEn', 'IncrEn', 'K2En', 'MSEn', 'PermEn', 'PhasEn', 'SampEn', 'SlopEn', 'SpecEn', 'SyDyEn', 'XApEn', 'XCondEn', 'XDistEn', 'XFuzzEn', 'XK2En', 'XMSEn', 'XPermEn', 'XSampEn', 'XSpecEn', 'cMSEn', 'cXMSEn', 'hMSEn', 'hXMSEn', 'rMSEn', 'rXMSEn']  
  
compare\_entropies()

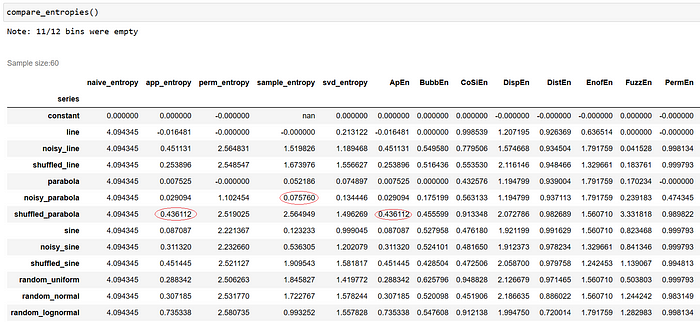


Figure 2. Comparison of entropies computed for each series. Produced by the author.

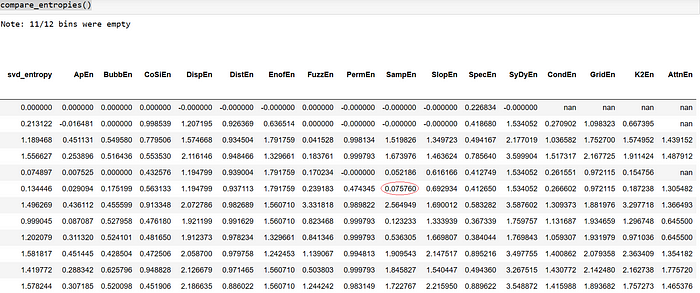


Figure 3. Rest of the comparison. Produced by the author.

As we can see, Sample and Approximate entropies (with default parameters) give similar results in both packages, while Permutation entropies don’t. It’s also immediately noticeable that a constant series, bearing almost no information in it, gets zero of nan entropy of any kind, and naive Shannon’s entropy is almost useless in selecting structured variables. Well, actually the formula I used was rather for a discrete variable, so no wonder.

Now, to the point of this small research. Obviously, the rest of entropies do vary quite a lot between the series. Can we spot the ones that take low values on pristine series, slightly higher values on noisy series, possibly even higher values on shuffled series, and the highest values on random series?

**Finding the most discriminating entropies**

Let’s first divide our series into subsets:

noisy = df.query('series.str.contains("noisy")')  
noisy.index = noisy.index.str.replace("noisy\_", "")  
  
shuffled = df.query('series.str.contains("shuffled")')  
shuffled.index = shuffled.index.str.replace("shuffled\_", "")  
  
  
rnd = df.query('series.str.contains("random")')  
rnd.index = shuffled.index.str.replace("random\_", "")  
  
normal = df[df.index.isin(["line", "parabola", "sine"])]

Then group by ratios between subsets and sort:

(  
 normal.div(noisy).mean(axis=0)  
 + normal.div(shuffled).mean(axis=0)  
 + normal.div(rnd).mean(axis=0)  
 + noisy.div(shuffled).mean(axis=0)  
 + noisy.div(rnd).mean(axis=0)  
 + shuffled.div(rnd).mean(axis=0)  
).sort\_values().head(6)

sample\_entropy 1.786956  
SampEn 1.786956  
app\_entropy 1.807307  
ApEn 1.807307  
K2En 2.309500  
svd\_entropy 2.698288

Let’s gaze at the winners:

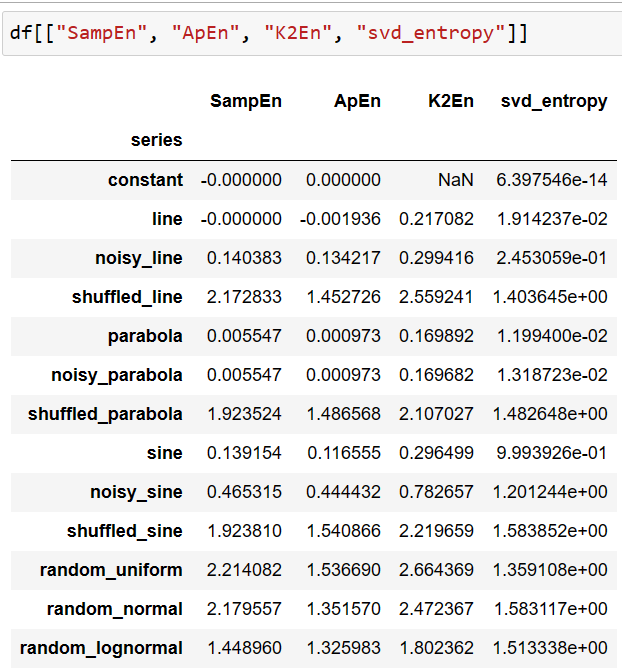


Figure 4. Entropies having the highest discriminative power. Produced by the author.

Indeed it seems like they are doing the job well! It was all not in vain. Perhaps you are not convinced and would like to proceed with higher number of samples?

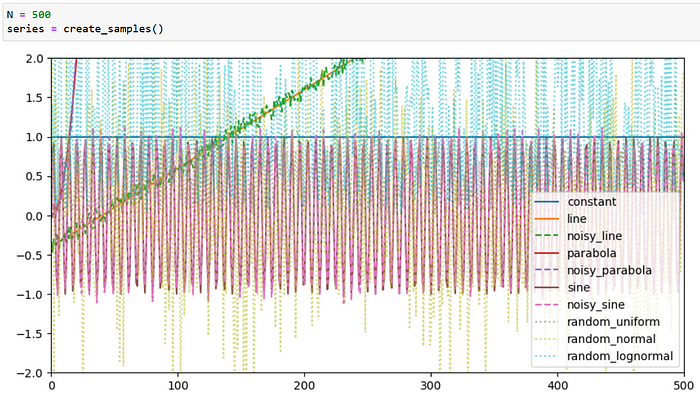
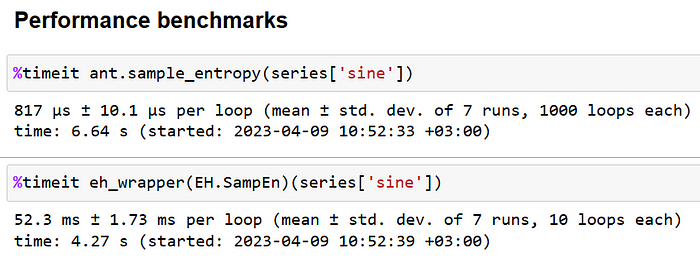


Figure 5. A teaser with more samples. Produced by the author.

I’ll leave that up to you by providing a [link](https://github.com/fingoldo/entropies) to the Jupyter notebook with complete source code. One last step before you go… It seems that EntropyHub has a much wider choice of methods, should we always prefer it? Not necessarily:



Antropy is much faster (as it’s using Numba under the hood), prefer it as long as it has what you need.

**Quick recap**

Entropies are a family of information-theoretic methods that, among other things, help differentiate between structured and unstructured variables, which can be useful in machine learning. There are two Python packages out there that can be leveraged to incorporate the power of entropies into your solutions.

*Thanks for reading this, I hope you’ve learnt something new today. If you speak Russian, welcome to my Telegram*[*channel*](https://t.me/AspiringDataScience)*where I occasionally post about data science.*