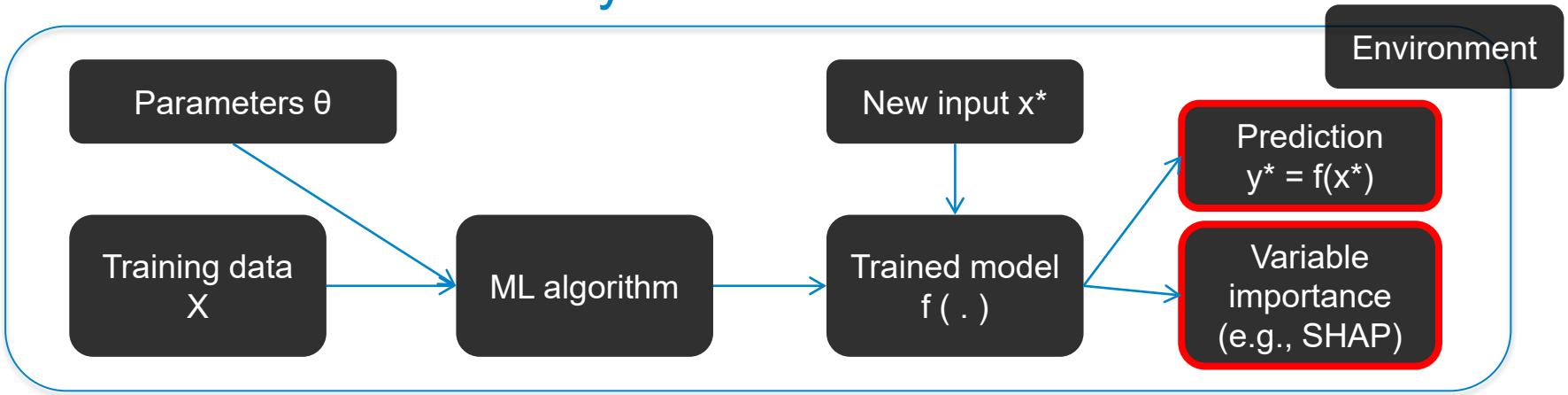
Ethics of Data Science – Part III

Week 2: Reproducibility and Robustness

Determinism and Stability

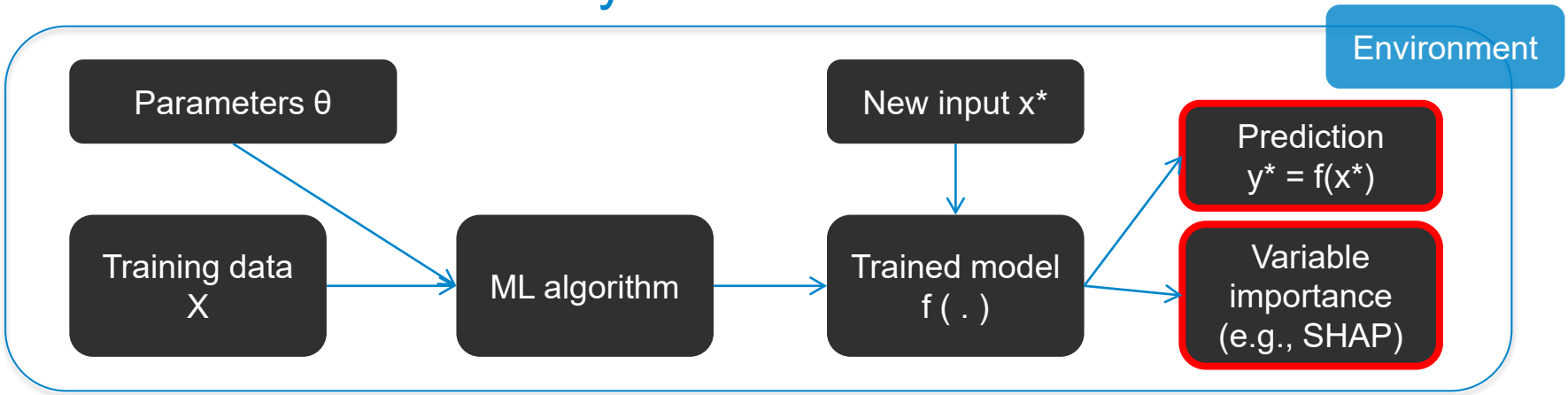
Dr. Chris Anagnostopoulos, Hon. Senior Lecturer

Determinism vs stability vs robustness



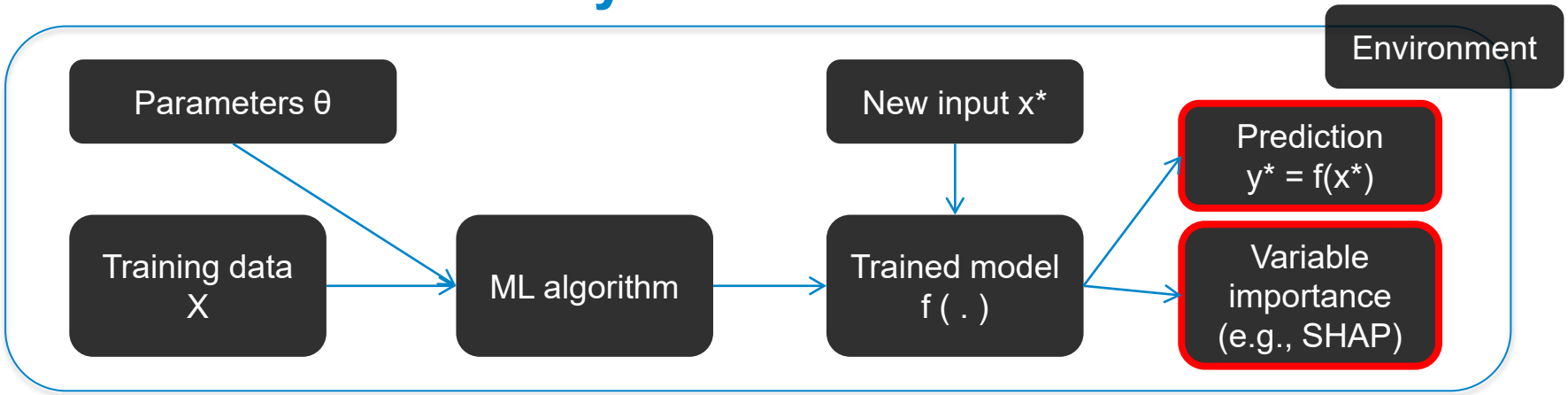
- **Determinism** is a literal interpretation of reproducibility: if you rerun with same input = same output.
- By “output” here we mean either the model predictions, or its XAI output (e.g., SHAP, PDPs, p-values, etc.). Note that mathematically, many XAI methods are statistical aggregates of predictions.

Determinism vs stability vs robustness



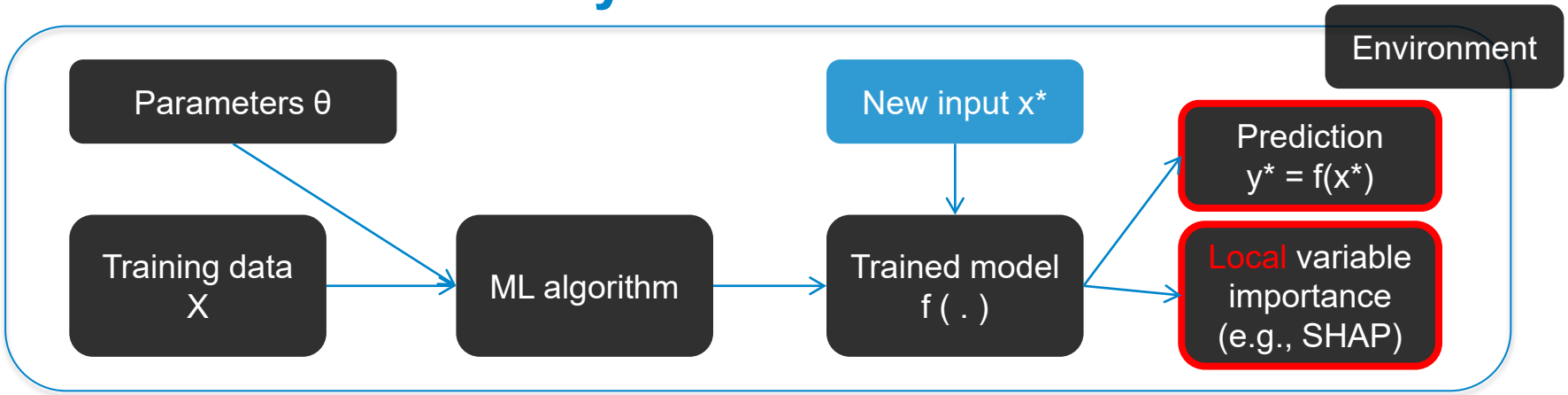
- A broader definition of determinism that is harder to satisfy is to insist that you get the same output when rerunning on same input, but possibly different environment (e.g., OS, number of nodes, etc.).
- We will see examples where the environment introduces tricky types of non-determinism.

Determinism vs **stability** vs robustness



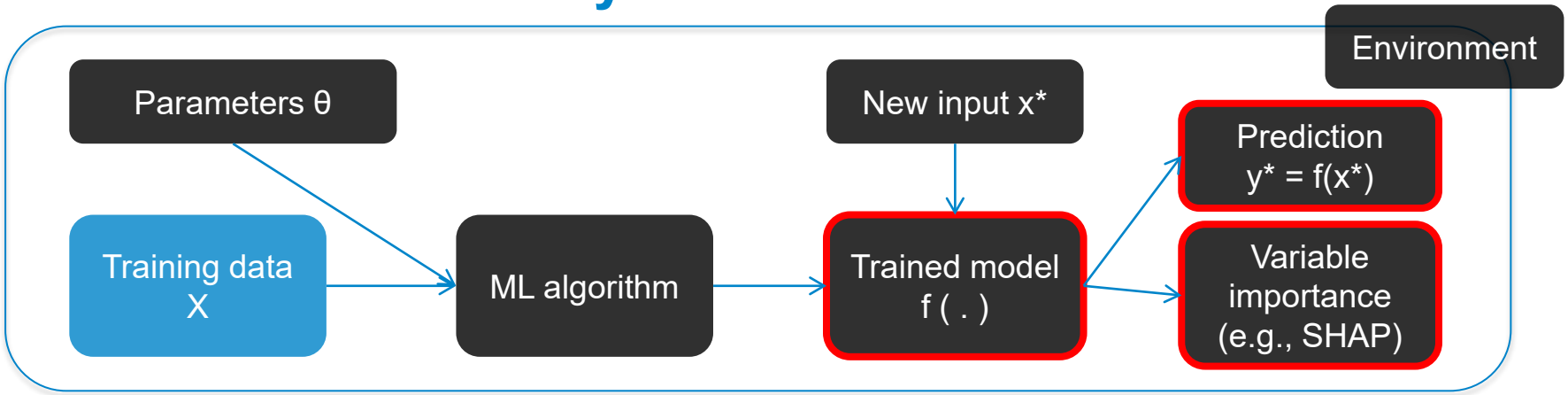
- **Stability** is a broader term that encompasses the idea that if the input changes by a tiny bit, the output also changes by a tiny bit. We can discuss stability of the training algorithm, or of the model.
- We can discuss the stability of a trained model, or of the training algorithm itself (the entire pipeline).

Determinism vs **stability** vs robustness



- **Stability of the model** occurs when similar input vectors produce similar predictions, and similar *local* explanations (global explanations do not offer any information on stability of the trained model).
- This type of stability is more similar to ”smoothness” and is not necessarily desirable (e.g., think of the effect that a single DNA mutation can have on the life expectancy of a human).

Determinism vs **stability** vs robustness



- **Stability of the training algorithm** (or estimation procedure, expressed in statistical terms) instead requires that if the training data is changed a little, the trained model changes only by a little.
- This a highly desirable property of a DS pipeline, and its absence can cause justified skepticism.

Determinism vs stability vs **robustness**

Robustness can be used in two different ways that apply to DS pipelines – which in fact overlap.

- **Robustness in software engineering** is about maintaining the ability to perform within given specifications in the presence of stresses (departure from ideal operating conditions), faulty inputs, or internal faults. So, for example, if you're a data scientist, you need to think about what happens if:
  - … your production environment has fewer computational nodes than you used in development.
  - … your production environment has some unstable nodes that tend to crash.
  - … your input data in production switch from encoding a missing value as "0" to "NA"
  - … your input data in production is erroneously populated from the user (e.g., they put in their age as 140 rather than 40 by accident, or write "forty" rather than the number "40")
  - … your model throws a numerical instability error in production (e.g., by trying to invert a matrix that is close to singular, whereas this never happened to take place during training).
- Many of these can be caught through data quality checks, but some faults may still reach the model.

Determinism vs stability vs **robustness**

Robustness can be used in two different ways that apply to DS pipelines, which actually overlap.

- **Statistical robustness** is instead about the ability of a model to generalize beyond the setting that it was initially designed for, i.e., beyond its assumptions about its inputs. This would include things like:
  - The ability of a regression model to deal with a small number of outliers / bad data points.
  - The ability of a model to maintain its predictive ability in the presence of a small amount of dataset shift (where the distribution of the input data is changing over time).

Determinism vs stability vs **robustness**

Robustness can be used in two different ways that apply to DS pipelines, which actually overlap.

- **Statistical robustness** is instead about the ability of a model to generalize beyond the setting that it was initially designed for, i.e., beyond its assumptions about its inputs. This would include things like:
  - The ability of a regression model to deal with a small number of outliers / bad data points.
  - The ability of a model to maintain its predictive ability in the presence of a small amount of dataset shift (where the distribution of the input data is changing over time).

- **Note**: the field of “robust statistics” started in the classical era of parametric statistics, where the model’s ”assumptions about its inputs” were transparently laid out. It is less clear what the assumptions of a model like XGBoost are, if any, and hence ”robustness” there can not be precisely defined. Instead, we talk about robustness to phenomena like outliers or dataset shift, i.e., noticeable departures of the test data from the distribution of the training data.

Determinism

How do I ensure or check that my model is deterministic?

Determinism

How do I ensure or check that my model is deterministic?

Same input + same code + possibly different environment = same output

1. Seed your random engine (or avoid using one altogether).
2. Avoid global variables (which act as an external state from the perspective of a function).
3. Ensure that the algorithm is always presented identical data, in identical order.
4. If you are working with distributed objects (e.g., a Spark DataFrame), check which operations are deterministic and which are not (e.g., operations that always return rows in the same order).
5. If you are working with distributed or multi-threaded computation, check which operations are invariant to the order in which different nodes/threads return an answer (concurrency-related).
6. Ensure your environment supports identical packages (e.g., via the use of Docker containers)

Determinism vs stability

Determinism can mask lack of stability.

Stability can mask lack of determinism.

Determinism is a reproducibility best practice.
Stability is a matter of validation and trust.

Fixing the random seed enforces determinism in a way that is very sensitive to the environment.

Mitigation

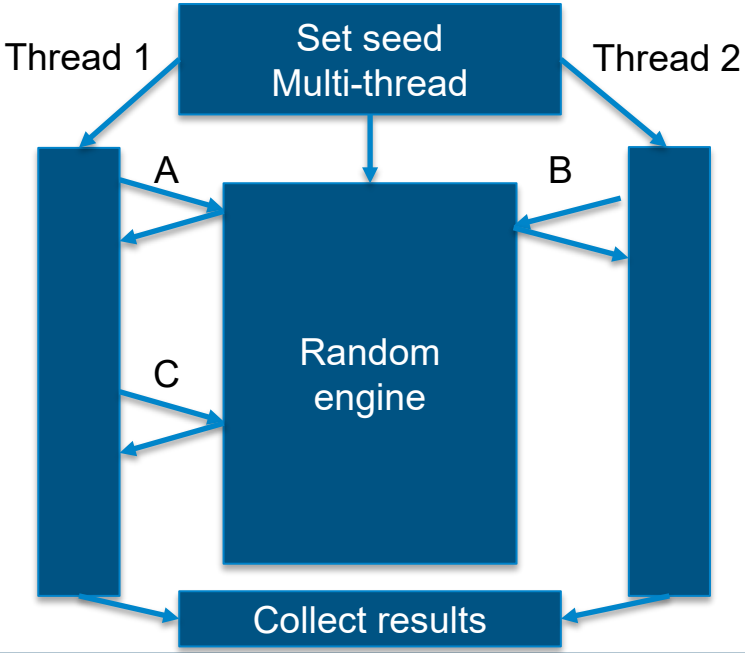
Test variability of results under bootstrap replicates

When confirming determinism, make sure to test for very small differences in many metrics or even the internal state of the model, not just top 10 important variables or aggregate stats like AUC.

Think of your audience and interpret what they mean by “stability” in any given situation.

Sometimes we may need to live with the lack of randomization to have the benefit of a more powerful, and indeed often more stable method.

Determinism: example with randomness and threads




```
>>> import random
>>> local_random1 = random.Random()
>>> local_random2 = random.Random()
>>> local_random1.seed(1234)
>>> local_random2.seed(1234)
>>> print(local_random1.randint(1,1000))
990
>>> print(local_random2.randint(1,1000))
990
```

Whenever compute is used in sophisticated ways (e.g., multi-threading, or distributed compute), ensuring determinism is hard, or even impossible.

Conclusion

- Determinism is about getting the same output when you have exactly the same input.
- Stability is about getting similar output when you have similar input, but in a DS context this should be interpreted as getting similar results (=XAI + predictions) when training on similar data.
- Robustness is about staying performant when your input or environment are being “weird”. Important examples of statistical “weirdness” are outliers, and dataset shift.

You can improve the determinism, stability and robustness of your pipelines significantly by a combination of an appropriate choice of model, data quality mitigations, and model fitting pipeline.