Easier AI

Simpler, smarter, faster, more flexible and understandable

Tim Menzies and the EZR gang *North Carolina State University*

Package: `https://pypi.org/project/ezr/0.1.0/`
Source: `http://github.com/timm/ezr`
Latex: `http://github.com/timm/ezr-tex`

We show how to simplify the process of AI analytics, which involves extracting high-quality insights from large quantities of data. We advocate for more efficient and accessible analytical methods that require fewer data samples and less complexity. This allows for easier verification and understanding of results. We highlights the benefits of using incremental methods in building models that can provide valuable insights with minimal data.

This work can be viewed as a (polite) protest against the prevailing preference for complex solutions in the industry, suggesting that simplicity could offer more practical and appreciable benefits but is often overlooked due to commercial interests. The call is for a shift towards simplicity in analytics, making it faster, smarter, and more flexible, to better serve practical needs and enhance comprehensibility.

Introduction

Suppose we want to use data to make policies– about what to do, what to avoid, what to do better, etc etc. How to do that?

This process is called *analytics*, i.e. the reduction of large amounts of low-quality data into tiny high-quality statements. Think of it like "finding the diamonds in the dust".

Many people have been doing data-driven analytics for decades. So it seems the right time to ask how can we make analytics simpler, smarter, faster, more flexible and more understandable?

For example, according to Tom Zimmermann (from Microsoft Research), there are many things we want to do with analytics:

	Past	Present	Future
Exploration Find important conditions.	**Trends** Quantifies how an artifact is changing. Useful for understanding the direction of a project. ▪ Regression analysis.	**Alerts** Reports unusual changes in artifacts when they happen. Helps users respond quickly to events. ▪ Anomaly detection.	**Forecasting** Predicts events based on current trends. Helps users make pro-active decisions. ▪ Extrapolation.
Analysis Explain conditions.	**Summarization** Succinctly characterizes key aspects of artifacts or groups of artifacts. Quickly maps artifacts to development activities or other project dimensions. ▪ Topic analysis.	**Overlays** Compares artifacts or development histories interactively. Helps establish guidelines. ▪ Correlation.	**Goals** Discovers how artifacts are changing with respect to goals. Provides assistance for planning. ▪ Root-cause analysis.
Experimentation Compare alternative conditions.	**Modeling** Characterizes normal development behavior. Facilitates learning from previous work. ▪ Machine learning.	**Benchmarking** Compares artifacts to established best practices. Helps with evaluation. ▪ Significance testing.	**Simulation** Tests decisions before making them. Helps when choosing between decision alternatives. ▪ What-if? analysis.

Note all the different algorithms in all the boxes. Before we knew better:

- We used to study those algorithms as separate things. Now we see them as very similar things, all of which call the same underling structure. This means that once we code one of them, we can quickly code up the rest.
- We 'd explore 100,000s of possibilities to find patterns in the data (e.g. during a "what-if" query). But these days, I can do the same analysis with

30 samples, or less[1] This means if someone wants to check my conclusions, they only need to review a few dozen samples. Such a review was impossible using prior methods since the reasoning was so complicated.

Why can I do things so easily? Well, based on three decades of work on analytics [1] (which includes the work of 20 Ph.D. students, hundreds of research papers and millions of dollars in research funding) I say:

- When building models, there are incremental methods that can find models after very few samples. - This is because the main message of most models is contained in just a few variables [1].

I'm not the first to say these things[2]. So it is a little strange that someone else has not offer something like this simpler synthesis. But maybe our culture prefers complex solutions:

> *Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better.*
> – Edsger W. Dijkstra

By making things harder than they need to be, companies can motivate the sale of intricate tools to clients who wished there was a simpler way. Well, maybe there is.

References

[1] TJ Menzies, M Dean, JL Black, and JF Fleming. Combining heuristics and simulation models: an expert system for the optimal management of pigs. In *AI'88: 2nd Australian Joint Artificial Intelligence Conference Adelaide, Australia, November 15–18, 1988 Proceedings 2*, pages 48–61. Springer, 1988.


```python
import re,ast
from typing import Any,Iterable,Callable
from fileinput import FileInput as file_or_stdin
#---------- ---------- ---------- ---------- ---------- ---------- ----------
def coerce(s:str) -> Any:
  "s is a int,float,bool, or a string"
  try: return ast.literal_eval(s) #
  except Exception:  return s

def csv(file=None) -> Iterable[Row]:
  "read from file or standard input"
  with file_or_stdin(file) as src:
    for line in src:
      line = re.sub(r'([\n\t\r"\    ]|#.*)', '', line) # kill comments,white space
      if line: yield [coerce(s.strip()) for s in line.split(",")]
#---------- ---------- ---------- ---------- ---------- ---------- ----------
class COLS(OBJ):
  """Turns a list of names into NUMs and SYMs columns. All columns are held in i.all.
  For convenience sake, some are also help in i.x,i.y (for indepedent, dependent cols)
  as well as i.klass (for the klass goal, if it exists)."""
  def __init__(i, names: list[str]):
    i.x, i.y, i.all, i.names, i.klass = [], [], [], names, None
    for at,txt in enumerate(names):
      a,z = txt[0], txt[-1] % first and last letter
      col = (NUM if a.isupper() else SYM)(at=at,txt=txt)
      i.all.append(col)
      if z != "X": # for cols we are not ignoring, maybe make then klass,x, or y
        (i.y if z in "!+-" else i.x).append(col)
        if z == "!": i.klass= col

  def add(i,row: Row) -> Row:
    "summarize a row into the NUMs and SYMs"
    [col.add(row[col.at]) for col in i.all if row[col.at] != "?"]
    return row
```

Listing 1: *Python example*

[1] Using semi-supervised multi-objective optimization via sequential model optimization (which is all described, later in this document).

[2] From Wikipedia: The manifold hypothesis posits that many high-dimensional data sets that occur in the real world actually lie along low-dimensional latent manifolds inside that high-dimensional space. As a consequence of the manifold hypothesis, many data sets that appear to initially require many variables to describe, can actually be described by a comparatively small number of variables, likened to the local coordinate system of the underlying manifold.