Easier AI

A programmer's guide to building simpler, smarter, faster, more flexible and understandable analytics.

Tim Menzies and the EZR gang North Carolina State University

May 14, 2024
`https://doi.org/10.5281/zenodo.11183059`

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

Analytics is the process of extracting high-quality insights from large quantities of data. Very simple and very fast analytics can be built by implemented by , combining, and simplifying many seemingly different analytics functions. These tools are "data-;ite"; i.e. they can reason extensively about complex problems using an incremental selection of just a few data samples. This "data-lite" approach. allows for easier verification and understanding of results.

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. We call for, when possible, a shift towards simplicity in analytics, making it faster, smarter, and more flexible, to better serve practical needs and enhance comprehensibility.

Audience

We write this book for programmers (or those that teach programmers). Here, we show the most we can do with AI, using the least amount of code.

In our own work, this material is used to teach a one semester graduate class in SE for AI.

Contents

1 Introduction

Suppose we want to use data to make decisions 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“.

At first glance, an analytics toolkit needs many functions. For example, in one survey of managers at Microsoft, researchers found nine kinds of analytics functions [1]. As shown in the following table, those functions include regression, topic analysis, anomaly detection, what-if analysis, etc:

	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.

But do all these seemingly different functions actually have a lot in common? Under the hood, are all these seemingly different things really just calls to a small number of things? And if that was true, does that mean:

- If we coded one thing, could rapidly code up many other analytic functions?
- If could optimize that small set of things, could we improve a wide range of analytic functions?

Well, our answers to these questions are yes, yes, yes, and yes. We’ve been working on application of analytics for decades [2, **?**, **?**, **?**] and supervised dozens of masters and 20 Ph.D. students (all publishing in the international research literature).

This document is a synthesis and a reverse engineering of the core engineering of analytics applications, After of work on analytics

Software engineers have a superpower that lets them simplify long lists of functions (like the above). That superpower is called *refactoring*, i.e. restructuring the source code so as to improve operation. This document applies refactoring to analytics. It will be seen that, under the hood, many analytics tasks share a similar set of underlying classes. This means that once we code one analytics function, then we can quickly code up many more.

For example, suppose we code a DATA class that stores rows of data. This class:

- Summarizes the columns of that data in NUMeric and SYMbolic classes (one for each column);
- Knows how to report the expected middle values of NUMs and SYMs (which is the mean or mode for NUMs or SYMs);
- Knows how to report the diversity about that middle value (which is standard deviation or entropy for NUMs or SYMs).

This DATA class offers most of the code needed to implement clustering and classification:

- A k-means clusterer picks centroids and random, then labels each row according to its nearest centroid. Those centroids are then moved to the middle of all rows with the same label and the process repeats. If all the rows with the same label are stored in a DATA class, then “moving the centroids” just means asking our NUMs and SYMs for their middle values.
- A Naive Bayes classifier keeps separate statistics for all the rows with the same classification. If each class is implemented by a DATA class, then all those statistics can be collected just by using the DATA code.

Better yet, once we have a clusterer and a classifier.

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 [2] (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 [2].

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.

[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.

```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"\    ]|#.*)', '', line) # no 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 independent, 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": # if 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*

I'm not the first to say these things[3]. 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.

[3] 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.

References

[1] Raymond PL Buse and Thomas Zimmermann. Information needs for software development analytics. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 987–996. IEEE, 2012.

[2] 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.