Last month’s column promised more on Naïve Bayesian Classification with a specific look at the performance with synthetically generated gemstone data from the little universe of James McCaffrey’s creation. But, deadlines being deadlines, life being life, and distractions and technological failings aplenty means that that promise will need to be deferred until next month. A new idea crept in and, because it was so apropos, it refused to get out until it was expressed in writing. And so, here we are, with an article on [leaky abstraction](https://en.wikipedia.org/wiki/Leaky_abstraction).

I hadn’t heard of this term until the end of August when I came across an article, written about [version control using Git](https://missing.csail.mit.edu/2020/version-control/), that mentioned it and then suddenly the light came on. Based on context, I basically figured out the general gist but it wasn’t until I actually did some research on the web and read the Wikipedia article that really knew how to talk about it. Here, finally, was a term for the angst I so often felt around the technology of the future – technology that was supposed to make things better and represent an improvement but which made life harder and more opaque; sometimes dramatically so.

A close up of a logo

Description automatically generated

The experience was a lot like suffering for years with a chronic condition, which you could not name or treat, only to visit a new doctor who not only believed something was wrong but knew what it was and how to diagnose it. Suddenly, you go from suspecting that it may all be in your head to being on the road to recovery or management with the mental relief of knowing you really were ailing the whole time.

In a nutshell, leaky abstraction is the term used to describe a common user experience with a tool whose interface makes the user experience an exercise in deducing how the development thinks about the problem their tool is addressing rather than using the solution they provide. While there is an even more formal definition, a set of examples comparing and contrasting (a la [Goofus and Gallant](https://en.wikipedia.org/wiki/Goofus_and_Gallant) in Highlights but with the opposite order) serves much better.

On the side of the angels is the modern automobile. One doesn’t need to know any physics or engineering to operate them. Starting the engine is effortless with no knowledge required of Newton’s law, mechanical advantage, fluid dynamics, thermodynamics, metallurgy or engineering principles. One doesn’t need to know if the brakes are direct mechanical or hydraulic, or are computer controlled to be able to stomp on them to stop quickly. The human-machine interface is intuitive. Want to go to the right, turn the steering wheel the same way. Want to go fast, push the gas pedal down. It is true that when things go amiss you are slightly better off if you know something about how the vehicle is designed but even in these relatively rare cases there is an entire support network built around making the process easy (albeit potentially expensive).

On the other side is Git. One needs to really know how Git’s data model works. The term ‘[directed acyclic graph](https://en.wikipedia.org/wiki/Directed_acyclic_graph)’ entered into my vocabulary because I am reliably told that the only way to really understand Git is to understand DAGs (e.g, the blog post [here](https://medium.com/hackernoon/understanding-git-branching-2662f5882f9)). But I am not interested in discrete math; I simply want to version control software. I appreciate it is a hard problem but so is designing a car or a laptop or a smart phone. Those systems move in the direction of simplicity but Git seems happy embracing a user experience that is best summarized by the following XKCD cartoon.

A screenshot of a cell phone

Description automatically generated

I’ve tried to like Git, really I have. When Github came online I was an early adopter precisely because I recognized the importance of saving to the cloud. I even am grandfathered in with a username that isn’t an email address. And I am grateful to Github for providing that platform. And I find Octocat to be a cute mascot (especially in his MegaMan guise).

A close up of a logo

Description automatically generated

But, weighed against all that benefit is the detest I have for the Git mechanics best exemplified by the biting part of the last word balloon.

Too many times I’ve been bitten by Git.

My first regretful experience came early on. By accident, I committed a large file that took my commit over the 50 MB limit Github had (perhaps it still does – don’t know don’t care). It was a careless mistake but I was tired doing real work. I only realized my mistake when I tried to push from the local repo to Github. No matter what hocus pocus I tried I couldn’t get Git to forget about that commit. Eventually I stumbled upon a magical incantation that remedied the situation but to this day I don’t know what it was.

Chagrinned by this incident, I tried to become better at Git but the arcana just left me cold. I’ve tried experimenting with Git under the argument that the best way to learn wasn’t to RTFM but to actually do. Along these lines, I created phantom project structures with files that are stubs and tried my best to move and reshuffle and rename and undo. One expert (see I was reading) argued that Git could track file moves with explicitly being told but it didn’t always work. Another expert said renaming would be a breeze. And so on. Numerous pieces of advice but in the end only vice.

A critical reader, especially of the haughty programmer types, are no doubt enraged with my lack of discipline to see an understanding of Git through to the end. But I would ask of them, do you understand the quantum mechanics that allows your beloved digital age to work. Judging by my experiences teaching the subject the answer is a resounding no. People, in general, and programmers, in particular, are bad at understand quantum mechanics. Suppose that to operate a computer one had to understand semiconductors, quantum wave functions, and Fermi-Dirac statistics (just to name a few). How many homes would boast a computer? Suppose you had to understand orbital mechanics and General Relativity to use a GPS-based navigation system. How many people would stay home (especially the haughty programmer types) for fear of getting lost.

No, I am neither lazy nor stupid. I am smart enough to know that software can do better that the leaky abstraction that Git foists on its user community. I am cautious enough to use the magical incantations precisely so that I don’t get bloated commits or detached heads or whatever. And I am industrious enough to put my time to better use than understanding the Git data model. If only I were as cute as Octocat I would be set.