Predictive Modeling: A Retrospective

Shreya Shankar

Jaunary 8, 2021

Introduction

"If I ever write a book before the age of 30, never talk to me again," I vaguely remember telling my friend something along these lines. But I am much younger than 30 and not yet ready to lose my friends. In no way is this meant to be a book.

There is merit to writing prose though, regardless of how young or old a person is. I find that it's helped me think critically about who I am and who I want to be as a programmer. Writing has always been a tool to manage my larger-than-average intensity of emotions. I am impressed by my technical peers and colleagues who seem to always make it through challenging work sprints in an enviably normal way. I feel like I scrape by.

I am quite an "emotional" programmer.[1] It's not like I throw tantrums while programming; I just feel strongly when I write code, see my results, and debug. Predictive modeling[2] as a discipline lends itself well to strong emotions – errors are often silent, projects often do not make it to production, intuition often trumps theoretical understanding, and more. As a result, my relationship with this field is somewhat complicated.

This essay is personal. It is a narrative about my experiences in and relationship with predictive modeling. It is a testament to the years I have spent banging my head against walls to derive value from predictive models. I am writing this more for therapeutic purposes than to share any great insights; in this, I do not intend to prescribe advice, and you will find many stories of embarrassing technical failures.[3] I hope you read this with a nonjudgmental frame of mind.

[1] I pay most attention to giddiness and dread. For example, when a system design change improves latency, I grin happily and do a victory dance in my head. On the other hand, when I have to search for and read through multiple log files, my brain plays a "power off" sound and I immediately check social media to procrastinate.

[2] I try not to use the catch-all phrase "data science" because there are both forecasting and analysis components to the field. This article has a nice overview. In my essay, I talk primarily about the predictive modeling aspect of data science.

[3] My intended audience is wide, including but not limited to:

- ML researchers who are curious about what applied ML is like
- ML practitioners who might appreciate consolation that they aren't the only ones making mistakes
- People generally curious about what it's like to work in applied ML
- Students or people interested in pursuing ML or data science careers

Childhood

I took my first algebra course in 7th grade. I do not remember much about the day-to-day experiences other than my bright pink TI-84 Plus Platinum Gold Silver Bronze 2.0 edition and Mrs. Park's "privacy fences" made of manilla folders that you could very obviously still look over and cheat if you wanted to. I played Block Dude on the graphing calculator a lot in that class.

One fine day, I decided I was done playing Block Dude. It bothered me that I did not know how to use many of the functionalities

on that calculator; the fanciest thing I could do was graph an equation of the form $y = mx + b$. [4] I stumbled across the `LinReg(ax+b)` feature and was instantly fascinated; if you entered a handful of data points, the calculator would find the line of best fit and give you a formula for the y value corresponding to any new x value! "This is called *extrapolation*," Mrs. Park said.

I took this extrapolation phenomenon and eagerly applied it to many aspects of my life. I constructed tuples of my previous 100 yard breaststroke times of the form `(time, age)` and input these into `LinReg`. *How fast will I be when I am 16.23 years old?* I wondered. The `LinReg` formula yielded 65 seconds, or a 1:05. Impressive. [5] Only many years later did I learn that the algorithm is not `LinReg`; it is something called linear regression.

Had I zoomed out on the graph, I would have noticed that the predicted times were negative after I hit 25 years old. But I was naive; I trusted my results as soon as I saw them.

[4] I think this concern is common among my friends who also took many systems courses. For a while, I felt more comfortable programming in C because I actually knew most of the language at the time. When my day-to-day consisted of more machine learning, I realized that proficiency in a language is how comfortable you feel implementing an idea in that language, not necessarily how much of the language you know.

[5] I never achieved this time.

Facebook

The summer after my freshman year, I interned at Google Street View. I spent more time in the B40 gym than my desk. I should have learned from this software engineering internship that I didn't like software engineering at a large company, but I apparently needed to repeat this experience to fully get it. At Facebook the following summer, my manager assigned me a full-stack project that I completed in 6 weeks but slowly released over the months so I would not have to do more software engineering.[6]

I am grateful for my manager at Facebook who noticed I finished early and told me that it's okay to not want to do another engineering project. At his suggestion, I asked a data scientist on the team if he had any project ideas. Fortunately, I worked on an interesting team (Civic Engagement), the company has "massive amounts of personal data" at their disposal, and that data scientist is awesome, so there were many intriguing project ideas. I settled on trying to predict whether a Facebook profile or page represents a US politician.

Armed with whatever knowledge I managed to obtain from Percy Liang and Chris Manning in two courses[7] (which I had taken as part of heavy[8] courseloads), I masqueraded as a data scientist for the last few weeks of my Facebook internship.[9] My data science colleague showed me how to `SELECT *` the data from relevant tables, randomly split the pages and profiles into train, validation, and test sets, and write a job to train and evaluate gradient-boosted decision trees. Facebook's infrastructure did the hyperparameter search; I literally only had to write a few lines of Python.

[6] I am a hare, not a tortoise. I sprint, then rest, then sprint, then rest. Strong emotions motivate me to either finish first or not work at all.

[7] Stanford's CS221 in Autumn 2016 and CS224N in Winter 2017

[8] I took 22 units a quarter that year. Really stupid in hindsight.

[9] I didn't know what SQL was, let alone whether to pronounce it as "sequel" or "S-Q-L." Thankfully my data science colleague worked in a different office, so we communicated mainly over Messenger.

Thinking back to this time, it's incredible that I could train models and not need to know about Hive, whether the data could fit in one machine's memory, anything about ETL or data pipelining, how config files were parsed, how jobs were scheduled, how machines magically had relevant dependencies installed, and more. I didn't even have to call `model.fit()` or know how decision trees work! I only needed to write my `SELECT *`, specify the fraction of data to allocate to train/validation/test splits, and indicate the type of model I wanted to train (GBDT). I can't claim that I was a curious intern and wanted to understand how all of this infrastructure came about; all I cared about was the results of the GBDT on my test set.

```
For work I am just running pipelines and finding features.
```

Figure 1: Journal entry for August 24, 2017

Since the infrastructure was nicely abstracted away from me, the biggest tool I had to improve the model was feature engineering (see figure 1). "We should train neural networks on profile text descriptions!" I excitedly suggested this feature to my data science coworker. I really hope he chuckled to himself at my total ineptitude before he kindly suggested we start with simpler feature ideas. Age, number of friends or followers, number of recent posts. I threw dozens of these simple features at the model and quickly became frustrated that I was getting diminishing returns.

My data science colleague did not seem fazed. He suggested a few more features – a binary indicator representing whether that person has a spouse and the ratio of number of followers to number of fans (people that "like" the page). These features had much greater importance in the trained model, and the trained model's accuracy boosted significantly. From this, I realized that good feature engineering is not about the quantity of feature ideas; it is about the quality of feature ideas. You want to come up with *discriminative* features, or features that have *unique* values for each class of data points. For example, number of fans alone may not be a good feature, since celebrity pages also have many fans. The ratio of number of followers to number of fans seemed was a better feature – I suppose people are more likely to follow and not "like" their politician's page than some celebrity's page.

The internship ended, and I gave a final presentation on the accuracy, precision, and recall scores obtained from various experiments and important features. I remember leaving feeling like I would miss my coworkers (see figure 2) and like I did something cool, that I did "real data science". I loved the rush of excitement I'd feel when an experiment produced "good" results or when one quality feature

proved to be more valuable than hundreds of useless features. Data science experimentation brought out high-variance, high-intensity emotions in me. I was addicted.

```
I feel terrible now.  I just left Facebook.  We took pictures.  This experience was definitely
so much better than last summer, but that means it's incredibly difficult to leave.  I learned
so much from everyone.  I feel like I have left a summer camp, but a camp where I learned A TON.
Literally.  I had so much personal and career growth this summer, unparalleled anywhere else.  I
feel like every season I gain 2x the experience of last season.  My growth is really catapulting.
It's exciting to be a part of it.  I loved the team.  12/10 experience.

I want to be a data scientist activist when I grow up.  I want to use data to inform my activism.
To do this I probably also need to take psych 1.
```

Figure 2: Journal entry for September 8, 2017

But I wish I had asked more questions in this experience. Why did I feel so accomplished after a data science project that I didn't productionize? Why didn't I strive to make my work useful? What precision and recall scores were good enough to be considered useful? How would I actually use the model to label Facebook profiles and pages? I suppose I should forgive myself for not thinking about these questions; for a student to continue work in a particular subject, it's so important for her to associate positive emotions with their first learning experience with respect to a particular subject. I was lucky to enjoy predictive modeling, and I had learned some valuable lessons from my time at Facebook – I learned to start with the lowest hanging fruit for feature ideas, thoughtfully craft discriminative features, and prioritize incremental iteration.

Google Brain

These days, I frequently talk about how machine learning in production is hard. Simply building a prototype of a machine learning model and showing that it works on a toy dataset, even if that data was collected from the "real world," doesn't mean this model can work for everyone at scale every day. I didn't realize this when I was at Brain, but now when I reflect, I recognize how much I learned then about the many challenges of real-world machine learning.

I will skip the story of how I ended up as a research intern at Google Brain as a result of dumb luck. I worked on adversarial examples and was directly supervised by two very kind, thoughtful, and intelligent researchers. I didn't do any machine learning work for any product at Google; however, my mentors both deeply cared about my personal growth and encouraged me to learn about other machine learning projects at Brain that could sparked my interest.

During my internship, a tweet[10] about biased translations from

[10] https://twitter.com/seyyedreza/status/935291317252493312?s=20

gender-neutral languages went viral. For example, Google translated "o bir doktor" and "o bir hemşire" in Turkish to "he is a doctor" and "she is a nurse" respectively in English. Fueled by my morals and (mainly) the anger from Twitter, I internally filed a bug in Google Translate. I suggested displaying gender-neutral pronouns when translating from a gender-neutral language.

I have really bad productivity right now. I can't focus. There was that tweet about Google Translate bias, and I decided to explore it myself. I picked a genderless language, Armenian, and translated "she is a doctor. He is a nurse" to it. Then I translated the Armenian equivalent back to English to get "he is a doctor. She is a nurse." She and he are the same in Armenian (no gendered pronouns), so even deciding on a gendered pronouns when translating is inaccurate.

I was fuming when I found this out. There are two problems at stake. One is the harder problem of bias in NLP, where the GLoVE or word2vec representations of words are inherently biased. There's a paper in which you can predict analogies like "doctor is to man as nurse is to blank" using GLoVE vectors, where the blank is woman. So clearly to solve this the training data for Google's seq2seq model needs to be augmented to favor underrepresented examples, but that would take forever to implement.

However, the above is not the problem I encountered. It's a larger relevant problem but not the problem I'm mad about in this moment. The problem is that Google's neural machine translator forces genderless pronouns to take a gender when translating. For example in Turkish, both he and she is "o." When translating "o" to English, NMT picks a gendered pronoun. Manually changing the probability threshold for the pronoun to pick will not work. Picking a pronoun in itself is incorrect. Maybe a quick fix to existing NMT translations from genderless to gendered is to append a layer to the end of the network that converts all gendered pronouns to "they" or something.

Anyways, I filed a bug to the Translate team reporting this inaccurate translation. Their response? "Gender is an ongoing research project" and they closed the bug. What the heck. They also linked to a "duplicate" bug about informal vs formal pronouns (like tú vs Usted, informal and formal "you" or आप vs तुम in Hindi). This "duplicate" bug was filed 4 years ago. I get that it's similar in the sense that you're trying to convert from a language that has no levels of formality to a language that does - which is like converting from a language with no gendered pronouns to a language with gendered pronouns. But this bug filed 4 years ago has not been fixed. Sigh. It is also filed as a "feature request" - when it is actually an incorrect translation.

I don't know what to respond with to the guy who closed the bug I filed. It is a bug, not a feature request. It has real implications. There is a simple fix to the small problem, and it opens the door to a larger problem of learned gender biases. I don't even know what to say anymore - I thought Google was the standard for inclusivity within customers, and not fixing this bug or being quick to dismiss it as a "gender problem" doesn't help.

Figure 3: Journal entry for October 5, 2017

Within a couple of days, the Translate team got back to me. They closed the ticket, saying they were investigating gender as an ongoing research project. *An ongoing research prjoect?* I furiously scoffed to myself and vented in my journal (see figure 3). *How long would this take to resolve?*

In 2019, Rachel Thomas confirmed that the same bug exists on Google Translate.[11] In April 2020, Google AI launched a pipeline

[11] https://twitter.com/math_rachel/status/1123354917404495872

to account for biased outputs in gender-neutral translations[12]. As shown in figure 4, Google Translate now shows two gender-specific pronouns in the output.

[12] https://ai.googleblog.com/2020/04/a-scalable-approach-to-reducing-gender.html

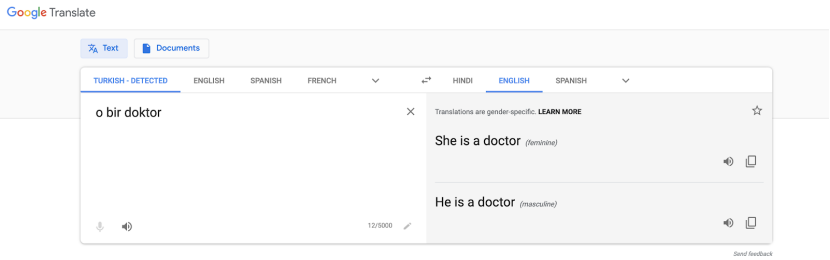

Figure 4: Turkish to English translation given by Google Translate in October 2020.

When I filed the bug, I was frustrated that no immediate action was taken. I judgmentally and incorrectly assumed the answer was simple. But now that I have actually worked in industry data science, I understand that the machine learning pipeline is much more complicated than the model itself.[13] Having visibility and a strong command of the end-to-end technical pipeline is a complex systems problem. Managing groups of people who have to effectively collaborate on these pipelines is also a complex systems problem.

I don't have much experience in ML bias or fairness[14], but I do now have experience shipping ML pipelines to production for clients to observe outputs from. When a client has a concern about a blatantly incorrect prediction corresponding to a certain data point, a feeling of dread washes over me. If the data point's feature values are within "common-sense" bounds (for example, age – a numeric feature – should be nonnegative), how do I possibly rationalize why the model's predicted output deviated significantly from the true label? Does this mispredicted example point to a larger phenomenon – poor performance for a certain subpopulation or addition of a new subpopulation that the model wasn't aware of – or do I just throw my hands in the air because models are imperfect and will sometimes make mistakes? And when I choose to throw my hands in the air, how do I explain why I made this choice to a bunch of people who weren't trained in machine learning but (understandably) expect ML pipelines to work like traditional software pipelines?

[13] Once we deploy a model, we struggle to "fix" bugs. For example, if model outputs are "biased," we jump to conclusions about how to "fix the bias" and patch it up as quickly as possible. Oftentimes this doesn't solve the core problem, which is usually bigger than just one misclassified output. In the particular example with "biased" outputs, the ML pipeline is developed and used in a way such that it systematically performs worse for certain subpopulations or subgroups in the data.

[14] There are many inspiring researchers doing great research in these areas.

Stanford

Many people dream about attending Stanford. I feel so lucky to have lived this dream. I loved the intellectual vibrancy in and out of the classroom, the people, the quirky events, my teachers, the weather –

the list is endless.[15]

I did not take too many machine learning or artificial intelligence courses in the latter half of my undergrad. After my Google Brain internship, I realized I wanted to become a better programmer, and I believed the best way to improve my programming skills was to write a lot of code. So I took operating systems, databases, compilers, and some other systems courses, and I somehow graduated from the systems track in Stanford Computer Science.[16]

I distinctly remember one machine learning course – deep reinforcement learning – in my junior year. At the time, I was also taking operating systems and strapped for time (see figure 5). For the RL final project, my group chose to implement an algorithm from an existing deep RL paper on a different dataset.[17] We were not able to replicate the paper's results or get the algorithm to work on our dataset. I even worked off a fork of the code released with the paper.

[15] Sometimes I forget that I live in San Francisco. Stanford was my home for so long.

[16] It is still a surreal feeling to have degree(s) from Stanford CS. I struggled through college. I genuinely did not believe I would make it.

[17] I will not name this paper, but it came from established authors from a reputable institution.

```
I'm getting worried that I have no idea what I want to do post graduation.  Startup?  Grad school?
Software engineering?  It's crazy; I don't know.  Maybe software engineering for a tiny company
where I can find good mentors?  I'm not sure.  Some days I feel as if the world is my oyster; other
days I feel as if there is only one path to success and I'm not even sure if I'm on it.  I feel
less confident about what I want to do post graduation than I was freshman year.  It's crazy.  But
maybe that's because I have so many opportunities, way more than I knew existed back then.

I am so behind in all my work.  I have not made too much progress in OS, which sucks.  There is
a lot of debugging left.  I have to implement 4 new syscalls.  Even the last assignment sucked -
I tried to implement the frame table for user virtual memory but page faulted on the main kernel
thread and spent days debugging this.  I really really cannot deal with systems.  This is by far
the most time-consuming class I've been in.

My group also hasn't even started the RL project.  I also have a multiple choice quiz and the DQN
assignment I have not started for that class.
```

Figure 5: Journal entry for March 5, 2018

Not knowing what to do, like many students who take Stanford AI Lab courses, I cherry-picked some good samples generated by the model to take up as much poster real estate as possible. My group mate added a large beautiful Q-learning diagram hastily made in Google Slides, not Ti*k*Z. The poster had at least 6 distinct colors. We got all the points.

Turns out we were not the only ones who slid our terrible results under the doormat before inviting others to our poster. During the poster session in the Tresidder Oak Lounge, my friend surveyed presenters in half of the room to find that 70% of the projects did not work, even though the posters indicated otherwise. People had all sorts of clever strategies to make good posters. One poster zoomed into a monotonically decreasing part of the loss curve instead of showing the full curve.[18] In another presentation, raw samples from

[18] When plotting loss over epoch or iteration, the y-axis scale can be meaningless if you don't have full familiarity of the data and training algorithm. You really only care that the loss decreases over time. It's more relevant to see a graph of the test set metric, such as accuracy, vs iteration.

the dataset covered at least a third of the poster. As usual, a large copy of the LSTM image from Chris Olah's iconic *Understanding LSTM Networks* [19] inevitably made its way to a handful of posters.

Now that I have my degree and Stanford (hopefully) cannot revoke it, I'm curious why this kind of culture exists in these AI classes. What are we trying to hide? Why are we trying to hide that it's hard to pursue deep learning ideas that work? Are we in denial that deep learning doesn't always work? Or are we just trying to get good grades? These classes have hundreds of students, eager for their deep neural networks to give them cool results. If left unchecked, such classes become divorced from reality and ladled with the expectation that most projects are cool and magically work. Students end up going into industry with an unhealthy dose of AI Saviorism.[20] Their industry data science projects fail. We all become disillusioned.

Modeling ideas that don't work are *not* anomalies. They are normal. The point of a class project is to try new things and learn, not produce "working" models. I wonder what the culture would be like if project TAs and course staff repeatedly communicated this to their students.

[19] Chris Olah. Understanding lstms, Aug 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[20] https://www.shreya-shankar.com/ai-saviorism

Viaduct

After finishing my undergrad, I joined Viaduct,[21] a startup that builds ML pipelines for vehicle OEMs,[22] as the first ML engineer.[23] I concurrently spent about a year finishing my computer science masters degree.[24] Since my masters degree was also in computer science (concentration in artificial intelligence), I had the lovely pleasure of simultaneously building simple models in industry to make money and faking complicated models in academia to get a degree.

Viaduct currently has production-running ML products, but when I joined Viaduct, we had no product. We had the leadership team, two kickass engineers, a data scientist who knew more about cars than the rest of us combined, and me.[25] As an ML engineer, I spent a lot of time doing both software engineering and data science, but in this essay I will speak only about my data science experiences. I *do not* talk about the following:

- Issues that arise from promoting models to production
- ML infrastructure
- Application performance tuning

[21] https://www.viaduct.ai

[22] Original Equipment Manufacturer

[23] I have a blog post describing the reasons I chose to work at a startup.

[24] Stanford has a coterminal masters program for undergrads; you typically spend 5 years and get both a BS and MS.

[25] In many applied AI companies though, the team itself can be the product. This can be confirmed during the acqui-hire they eventually go through. Viaduct is cool because it does not have this ambition to get acqui-hired; they are building things to deliver value.

"Just do something interesting with the data"

I had way too much agency for my first ML project at Viaduct. The task was to use ML to do "something interesting." In hindsight, this is not a good task definition because ML is only a tool – the emphasis should be on finding a business-relevant problem to solve. But at the time, our only paying client didn't care too much about what we built; they just gave us a year of sensor data from a set of vehicles for us to play around with. I'm not even sure if they expected anything. It was probably some R&D project for them.

An engineer colleague helped me set up AWS CLI on my work computer and showed me an S3 path pointing to where the client data lived. The path matched the pattern `s3://scratch-etl-{something}/*`. "What's ETL?" I asked him. He happily explained the extract-transform-load [26] data paradigm to me.[27] I did not know what S3 was, but I had heard of data "buckets" before.[28] The engineers had written some library functions built on `boto3` to connect to the relevant data, and these functions (as well as seemingly everything else they wrote) worked well out-of-the-box. Now all I had to do was write a Python script to "do something interesting."

My boss guided me towards something potentially interesting: what did an "anomalous" drive mean and look like? If a good model believed sensor data for a drive was sufficiently different than what it had seen for that car in the past, maybe the vehicle owner would like to be alerted. I treated this project like any ML project I had worked on in the past – first I read some papers about deep learning-based anomaly detection methods. Next, I preprocessed the data by selecting numerical features and normalizing them. Then, I used TensorFlow to write a small autoencoder with only fully connected layers that minimized the mean squared error [29] (MSE). Finally, I kicked off the script to train a model, only to find that my CPU took forever for one epoch to train.

It had been a week, and I had no results to show![30] Frustrated, I asked the engineers if I could train my model on a GPU. I imagined there was some virtual machine I could SSH into, `scp` my project files over, and run the code. My coworker patiently explained to me that this was not the case; this is not the case at most companies. "If you want to deploy your training job on a machine with a GPU, you can use Argo,[31] which helps us create workflows on Kubernetes," he told me. He showed me a template workflow he had written for deployment on the GPU cluster and how to use this template for my job.[32]

The training job finished, and I got similar train and validation split MSEs. That seemed like a good sign; I was not overfitting. The

[26] Wikipedia contributors. Extract, transform, load — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=Extract,_transform,_load&oldid=998138895. [Online; accessed 5-January-2021]

[27] He is too nice to have gone back to his desk wondering what kind of first ML engineer hire doesn't know what ETL is.

[28] I imagine there are gigantic buckets sitting in fields in the middle of Oregon or wherever `US-West` clusters are located. Like windmills, massive and spaced apart, only instead of rotating fans, the buckets contain large amounts of 0s and 1s.

[29] Wikipedia contributors. Mean squared error — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=995577263. [Online; accessed 5-January-2021]

[30] It might sound silly to expect any new employee to produce something in a week, but I was used to completing projects quickly from the AI classes I had taken.

[31] https://argoproj.github.io

[32] I had never used containers or Kubernetes before. Viaduct exposed me to this whole new world of infrastructure.

next natural step to me was improving the model. My data science colleague had mentioned he was using temporal convolutional networks [33] (TCNs) in his project, and I was curious to see if TCNs would boost my results. Sure enough, the validation MSE dropped, and I excitedly reported to my coworkers that something worked.

```
Todo:  write an autoencoder that...works?  Lol
```

Figure 6: Journal entry for July 11, 2019

But did it really work? I didn't know what to make of an MSE of 0.1. What this "good enough?" For what purpose? Businesses care about metrics in terms of dollars, not raw MSE values. Fortunately or unfortunately, I let go of this project the next day to focus more important project.

Personalization tools

My next project goal was to identify drivers' home and workplace locations based only on de-identified vehicle sensor information for drivers who had opted in to a specific program that we had partnered with. In hindsight, this project also did not seem to have a clear map from model accuracy to dollars saved. However, I was still high on the new freedom and flexibility gained from working at a very small early-stage tech startup.

I charged forward with my TensorFlow scripts and newfound `matplotlib` prowess to apply deep learning methods for clustering. However, once the model output its results, I struggled to validate them. For a handful of vehicle IDs, I entered their most prominent cluster latitudes and longitudes into Google Maps. I clicked on the satellite view. Could I see a house? After finding that the first 10 vehicle IDs' top clusters pointed to the middle of highways, I determined that my model was useless.

What next? Finally, I decided to try something other than deep learning.[34] I used `scikit-learn` to fit DBSCAN [35] to the data and went through a similar process to validate the top clusters. At least this time around, 4 of the 10 clusters pointed to actual homes. But I was assuming that the largest cluster corresponded to a person's home, which might be incorrect for some people. I also would have to figure out how to identify workplace locations.

Then I implemented the stupidest algorithm I could think of: identify the most frequent location for each vehicle at 2AM. Again, I checked each location in Google Maps. 10 out of the first 10 locations pointed to homes. I checked around 50 locations, and only 3 or 4 did not point to homes. I didn't know whether to feel depressed or excited.[36]

[33] Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks: A unified approach to action segmentation. *CoRR*, abs/1608.08242, 2016. URL http://arxiv.org/abs/1608.08242

[34] I don't know why I was so obsessed with trying to make deep learning work. Good problem-solving is about finding the simplest solution that meets the requirements.

[35] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996

[36] I think my AI Saviorism began to break down around this time. I felt some tension that I wasn't learning anything new in ML, since I had to resort to simple methods. But I was learning things, practical things, relevant things. I was learning about what actually delivered value in the "real world."

Data cleaning

For the last 14 months or so, I've worked primarily on predictive maintenance for clients. The problem is roughly: for a given component in a vehicle, what is the probability of failure in the next month?[37] Predictive modeling on time series data at scale is really, really hard.

In my first few data science projects at Viaduct, the "raw data" that clients had given us was actually cleaned by their data scientists. I did not know this at the time; I never thought to ask. One day, we got a new client – they went through our standard procedure to give us access to their "raw data", and an engineer coworker gave me the relevant paths. I quickly coded up some deep learning architecture to minimize binary cross-entropy loss. The precision and recall scores were atrocious, worse than random guessing.[38] Without investigating much further, I attributed the issue to deep learning problems. It was not because of deep learning,[39] but I am nonetheless glad I came to that incorrect conclusion, otherwise I would never have made a commitment to stay away from deep learning unless deep learning was absolutely necessary to solve the problem.

When logistic regression did not produce results, I finally inspected the data.[40] I was shocked, or "shook" as a good Gen-Z person would say, to find vehicles that had driven over a *billion* miles. Some vehicles had used a negative number of gallons of gasoline in the last few months. The data clearly had errors, and I was excited to find some low-hanging fruit that could improve the models.

In our first pass of data cleaning, we hardcoded lower and upper bounds for prominent sensors. However, some clients had given us access to data from over 100 sensors, and I, who did not know a single thing about cars, had no idea what some of the sensors measured. How was I supposed to figure out what an "outlier" meant? I used equation (1) to define "outliers" by identifying loose bounds for each feature.

$$\eta_{.50} \pm \lambda(\eta_{.99} - \eta_{.01}) \tag{1}$$

where:

η_p = p * 100th percentile in the distribution[41]
λ = some multiplier, 5 or 10 in my case

The first time I generated the bounds, I clamped outliers, or set them equal to the nearest bound. However, many times these bounds themselves were nonsensically valued – for example, some vehicles had 10 million miles instead of 1 billion miles. Since this was time series data, a coworker explained that I could break down "bad data

[37] Month is an arbitrary unit of time here; different clients care about different windows of time.

[38] Sometimes modeling – deep learning, particularly – is especially frustrating to me because most of my failures are silent. My code passes through the compiler fine. I hardly get runtime errors. I usually know something is wrong because my results are awful, and I don't know where to begin debugging. This realization gave me a new appreciation for software engineering, since my code usually only executes successfully if and only if I have no errors.

[39] Until this point, I had normalized my data by subtracting the mean and dividing by the standard deviation. But in the "real world," data is not necessarily normally distributed. I switched to min-max scaling and got better results.

[40] This is the first thing any data scientist should do. I can take comfort in the fact that I was hired as an engineer, not a data scientist. It is okay that I didn't know how to be a great data scientist in my first few months after undergrad.

points" into two categories: outliers that should be clamped, and outliers that should be treated like "missing" data points or forward-filled with clean values. I removed the clamping logic and set the outliers to `null` in upstream tables. I automated this logic as part of a data cleaning pipeline.

About 8 months later, I found a large number of vehicles with a bunch of `null`-valued number of engine revolutions. Turns out that tight bounds need to become looser over time! The total number of miles driven will increase! The total number of engine revolutions will increase! I reran the outlier analysis on raw sensor data to produce updated bounds. But the models we had running in production were trained on sensor data within tighter bounds. What models did I need to retrain? Whose models did I need to retrain? I found dread creeping into my body again – one small change upstream can have hundreds of silent negative downstream impacts.

I once used the label as a feature

The subheading says it all. It is quite embarrassing.

Once, a client sent us an email, asking us to train models for a different component before the next day. Since this was a different prediction task, I had to write new code to generate new labels to add to our feature table. I ran my old modeling code, which essentially did the following:

```
df = load_data(...)
feature_columns = list(df.columns).copy()
feature_columns.drop('days_until_failure')

# Other code
...

# Preprocess data and train model
X = df[feature_columns]
y = convert_to_label(df['days_until_failure'])
...
```

Pressed for time, I scrolled to the bottom of the file, changed the line

```
y = convert_to_label(df['days_until_failure'])
```

to

```
y = convert_to_label(df['new_days_until_failure'])
```

and quickly ran the offline training and evaluation jobs.[42] The met-

[42] The labels were actually named in a more descriptive way; I just use `days_until_failure` and `new_days_until_failure` to avoid disclosing relevant business details.

rics were not perfect, because there is a nontrivial transformation computed in the `convert_to_label` function. But still, obviously this gave me great results, and I reported them to the client.

The next day, due to dumb luck and sheer curiosity, I decided to inspect this particular model. What were its top features? I looked into SHAP [43] results and was horrified to find that the top feature was `new_days_until_failure`. How could I have been *so* stupid?

I always believe that when you make a mistake, you should let everyone know as soon as you find out. So I told my colleagues, expecting them to also be horrified. I think they were, but they forgot about it soon afterwards because they had many other things to do. The client also conveniently forgot about this request.[44] So it ended up being okay, but I was determined to never make this mistake again.

"A more robust solution to avoid this mistake is to require whitelisting of features," my colleague mentioned. I totally agree. He modified our modeling APIs to require a list of feature names as a parameter. It might be more tedious for a data scientist to explicitly list hundreds or thousands of column names as features they want to use, but at least with this solution, they are much less likely to use the label as a feature.

[43] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2(1):56–67, Jan 2020. ISSN 2522-5839. DOI: 10.1038/s42256-019-0138-9. URL https://doi.org/10.1038/s42256-019-0138-9

[44] I suppose when clients make hasty requests that aren't in the specified deliverables, they don't care too much. But I cared so deeply; I was so worried that this reflected poorly on my competence.

Different modes of experimentation

Over the years, I learned that I have two different "modes" of data science. In one state or version of myself, I'm experimenting as quickly as possible. I have a list of ideas, all of which can be tested with few-line code changes, and I come up with more questions and ideas every time I cross something off on that list of ideas. I can experiment endlessly in this mode. I load a subset of data into a Jupyter notebook and hackily mess around to learn more about what's going on. Maybe I add features or change the model or change the data. I just want to get feedback quickly, and fast feedback is *so* important to feel productive while iterating on ideas.

This mode can impress other people, but there is a dead end. Because I care so much about iterating quickly in this mode, I care very little about reproducibility. It is a skill I've developed over implementing hundreds of data science project ideas. But to make a data science project *useful*, you need to also be able to reproduce results of any experiment you run. Endless Jupyter experimentation is not reproducible. For a basic example, consider the following code in a notebook format:

```
# Cell 1: imports
import random
```

```
# Cell 2: set seed
random.seed(0)

# Cell 3: generate random number and do other things
num = random.randint(0, 10)
...
```

I would probably only run cells 1 and 2 once, as soon as I opened the notebook and started the kernel. The first time I run cell 3, `num` will be set to 6, and other relevant code will execute. Suppose I change the other code and want to rerun cell 3. The value of `num` changes! Even genuine attempts to ensure reproducibility, like setting a random seed, can get lost, mainly because the graph of computation performed in "endless experimentation mode" can be different from the graph of computation performed if you run the cells once, top to bottom. As Joel Grus says, "notebooks have tons and tons of hidden state that's easy to screw up and difficult to reason about."[45] People across the world of various skill levels have mixed strong opinions on notebooks. Notebooks are probably not going away, even if they have problems.[46]

So I learned the second mode, the slow mode, when I needed to show results to other people. I deploy the pipeline with the DAG scheduler, version everything – data, model, code, artifacts, you name it – and see if I can replicate results multiple times through cross validation. I never trust my results or communicate them to someone else until I've seen results of the second mode. I am rigorous in this mode; I put my software engineering hat on and get to work. It is tedious, but if there is anything I have learned about data science, it is that I need to be disciplined and patient in order to repeatedly reap the benefits of this sorcery.

[45] Joel Grus has an excellent relevant presentation on this.

[46] Maybe the way to fix Jupyter to avoid hidden state problems is to reset state whenever a cell is changed to what it was immediately before that cell was run. You could free relevant memory when a cell is deleted. But there are a lot of edge cases here; this solution would not work all the time and could be more of a hassle.

I learned Spark

Good data scientists are expert gymnasts. They know how to wrangle and twist data into random formats, data structures, and pipelines – all in multiple languages.[47] A month or so into my job, I needed to learn Spark to write some of the first ETL pipelines for our company.[48] I grew to love Spark. It is a rabbit hole you can continuously go deeper into. I think a fun language consists of two things: it is easy to quickly pick up a working knowledge of, and you can constantly learn new things and feel smart when you learn them.

When we got our first dose of "big data" that wouldn't fit on one machine, we started using Spark to generate features, dump them in a table, then subsample this feature table into relevant training

[47] I thought I had to become a gymnast after obscene pointer gymnastics in my systems courses, but this is nothing compared to all the obscure `pandas` manipulations my former data scientist colleagues know how to do.

[48] Thankfully, learning Spark was not so bad, since I had taken a databases course in college. I like learning new programming languages and frameworks – there is a beautiful frustration that comes from having a thought in your head and not being able to express it in the particular language. It is a wonderful to watch this feeling slowly fade over time as you learn the language.

files that could fit in memory. At the time, a data scientist and I were the only people building machine learning models. When we had to work on the same modeling problem or prediction task, we did not anticipate how many technical issues we would run into as a result of dividing up the computation. We just divided the list of features in half, individually wrote code to compute each feature, and output our features into tables of the same schema. Our individual Spark feature generation pipelines read from the same upstream tables of data; however, our resulting feature tables had different numbers of rows. How does one even go about debugging this? The pipelines were complicated, and we write code differently.

I spent a day trying to identify the differences but didn't come up with anything. We had a deliverable coming up, so I just moved forward to join the tables and train a model. The number of rows weren't all that different; maybe we could still get some results.[49] I had not specified a primary key in constructing the tables, but implicitly I knew that the primary key was a compound key, consisting of "vehicle ID" and "repair date" columns. I joined the tables on the relevant keys without considering the fact that *not all vehicles have a repair date*, thus some "repair date" values were `null`! The join messed up the expected number of rows in the resulting table, I ended up getting double the number of positives, and my modeling results were inflated.[50]

So I rewrote the pipeline from scratch, adding `assert` statements after every join to make sure the number of rows in the resulting table didn't change. Later on, after a coworker redesigned the pipeline to generate a versioned[51] feature table on a schedule, we also added `assert`ions to check for unique primary keys. I love `assert`ing when I am writing Spark – my goal is to restrict as many possibilities for silent failure to the modeling stage, not ETL or feature generation.

Is data science an art or a science?

Some people say data science is more of an art than a science. Donald Knuth famously says "science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it...computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty." [52] I can get meta and talk about how art and science are closely intertwined; both require careful, methodical thought and execution as well as a certain amount of creativity to construct something no one has made before.

In traditional software programming, you, a human, produce code

[49] This is bad engineering. Very bad. I have never done this again.

[50] On the bright side, this problem highlighted an unexpected difference between the feature tables. Our "repair date" values didn't match up.

[51] This is the only MLOps rant I have (see my Twitter for many other rants): the process of iterating on good infrastructure for ML can feel endless. We didn't version our data, models, and other artifacts for almost half a year. In hindsight, it seems silly to set up an ML experimentation pipeline without versioning. There's no other way to ensure reproducibility. But as soon as we had a product and many people working on the same ML problem, the need for versioning was apparent, and we set this up fast. Post-versioning, initially there was a pain point of tracing a result back to all of the versions of the components that produced it – you need to find the Airflow run ID that produced it, code outputs or logs, associated metrics, git hash, and more. Once we started deploying models in production, I realized I needed infrastructure to "trace" our production ML pipelines – both the specific model binary as well as the lineage of models over time in that pipeline.

[52] Donald E. Knuth. Computer programming as an art. *Commun. ACM*, 17(12):667–673, December 1974. ISSN 0001-0782. DOI: 10.1145/361604.361612. URL https://doi.org/10.1145/361604.361612

to solve a problem. When doing predictive modeling – an important facet of data science work – you, a human, produce code *to produce code* to solve a problem. This kind of data science work needs creativity in many stages. Creativity is necessary in feature generation – the biggest gains I've gotten in data science come from "genius" feature ideas, particularly those suggested by domain experts.[53] Creativity is also required in the actual act of modeling, which can sometimes frustratingly feel akin to compiler optimization in hopes that the resulting generated binary does something different.[54]

At Viaduct, I had the privilege of thinking about the same ML problem for over a year. As expected, I had more ideas on my mind than time to implement them.[55] Ideas ranged from things like running gradient boosting on different batches rather than the same dataframe every round to optimizing different objectives. The people I spend time with also inadvertently influence my ideas – for instance, I had (and still have) a huge crush on someone who does research in deep reinforcement learning, and for a while, I thought about using RL to compute a policy of when to fix at-risk vehicles given certain constraints to minimize total on-the-road failures.

I experimented with a new modeling idea once a month, on average. When my colleague or I had a new idea, we'd immediately DM the other person and spend hours together, prototyping, coding, experimenting, and learning.[56] Most of our ideas could be implemented and tested in a couple of long programming sessions, but we didn't shy away from larger challenges. For example, in one large project that spanned several weeks, we trained Transformer [57] models on raw sensor data to produce embeddings to use in downstream models. The engineering challenges around dataset size and integrating this into a production pipeline were massive, but the most excited I ever felt at Viaduct was when I saw the Transformer working – for each sensor, it predicted the next time step's value better than forward filling from the current time step's value.[58]

Many ML practitioners claim you do not need to feel confident in your mathematical abilities to apply ML well. I disagree. I needed mathematical maturity to understand the science of existing survival analysis modeling techniques[59] and pursue the art of experimenting with related ideas. Working with my programming partner coworker was especially great for me because he actually studied math in college, and math is definitely not my strong suit.[60] In one fun modeling idea experiment, we constructed a likelihood function of the exponential probability density function, as shown in equation (2), and wrote a custom `xgboost` loss function to maximize it. To write a custom `xgboost` loss function, you need to compute the first and second order gradients with respect to the function that the model

[53] I've learned that domain experts don't necessarily know machine learning or relevant terminology. To get good feature ideas, I don't explicitly ask for feature ideas. I ask them how they would go about making a prediction on a raw data point – what's the first thing they would look at?

[54] I heard this compiler optimization analogy from Dylan Hadfield-Menell.

[55] For each problem in my life, I have a process allocated to think about it. All of these processes are either running in the background or foreground. They are *always* running. It really sucks. Sometimes I wish I could just send a SIGSTP to a process corresponding to a problem I can't solve and a SIGCONT when I am ready to think about it again. Unfortunately, I am only capable of spawning new processes and sending SIGKILLs.

[56] I cannot understate the value of having a good programming partner to implement ideas with. It makes the experimentation process actually fun, not just bearable. Most people feel more productive programming alone; I certainly felt this way before I met one of my coworkers. I am significantly faster and feel more powerful when I am working with him.

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762

[58] Why deep learning? In this case, we had around a terabyte of sensor data that we didn't know how to fully leverage as features.

[59] There are several models in this family, such as Cox Proportional-Hazards or accelerated failure time (AFT).

[60] I'm easily the worst mathematician in my college friend group, and I even took around 5 math courses that weren't required for the CS major. I wonder if most Stanford CS grads also feel weak at math.

is trying to learn ($\lambda(x)$ in this case). This is not a machine learning paper for me to flex that I can compute gradients and Hessians, so I will spare you from the computation; after we coded up the loss function and ran the experiment, we unfortunately had to add it to our mental list of ideas that didn't work at the time.[61]

$$L(t) = \lambda(x) \exp(-\lambda(x) t) \tag{2}$$

[61] I have many failed ideas. I tell myself I do not write them down on paper because I want to experiment faster, but it is probably because I am lazy. This is probably not a good strategy in the long term.

I learned that in data science, the goal is not to minimize the number of failed ideas; it is to accept them as part of the process. I find it more useful to think about data science not as an art or a science, but as an emotional relationship with the training algorithm. I come to the relationship with a problem, and I ask the training algorithm to produce a model. It usually runs but doesn't give the results I want on the first try. I have to be patient and sympathetic to both what the training algorithm can do and how quickly or slowly I learn its quirks. When something doesn't work the way I expected, do I blame myself or the training algorithm? Over time, I realized that the blame doesn't matter. We are both partners in making a machine learning pipeline deliver results. This relationship is still a work in progress for me, but I am slowly learning to accept limitations, let go of unrealistic expectations, and allow myself to be pleasantly surprised when something works. Ultimately, the goal is to reach a level of intimacy with the algorithms such that words alone cannot romanticize it enough; the results continually speak for themselves, and everyone understands that the results could have only come from the programmer's unique symbiotic relationship with their machine.

Product-driven data science

When I helped release a product built around my predictive models, we experienced a jarring 20% performance drop between the metric on the offline evaluation set and the metric on live predictions. There's no way for me to know "exactly what all the issues are," but one issue I observed was label lag – we learned about vehicle component failures sometimes months after they occurred, so our metrics on our live predictions weren't always up to date. A plethora of other problems exist when delivering a product around predictive modeling, but I will not talk about those here.

At the end of the day, businesses just want to make money. To provide business value, I had to work with nontechnical stakeholders that understood exactly how the money flowed in and out. I didn't know what problems cost our clients the most money. My nontechnical coworkers understood these problems deeply, much better than I did. From working closely with them, I learned several things:

1. Promising data science results is hard – we often don't know the upper bound of ML performance. Maybe the data is just impossible to separate in $\mathbb{R}^n$. We don't know things about high dimensional spaces; we just pretend we do.

2. Solving the hardest ML problem looks impressive, but solving a problem that saves the most money is actually the most impressive. These two problems are not always the same – for example, when you have an imbalanced dataset, achieving high recall could be easier than high precision.

3. Baselines aren't necessarily machine learning models; they are whatever solution currently exists. They could be rule-based. They could be nonexistent.

4. Writing a production machine learning pipeline is mainly software engineering. Separate inference pipelines from evaluation pipelines, since you may run them and publish results at different times. Make sure your computation graph is a DAG, not a cycle, especially when you inevitably add new tasks. Take time to plan out the data models. The schema for the tables directly powering the product inevitably change as you are iterating on the product, so design data models such that you have to change schema for as few tables as possible.

5. Debugging models in production can lead to frustration and exasperation when focusing on a few individual mispredicted data points. It's hard to draw any general conclusions from such debugging investigations.

6. When you build predictive models that humans make decisions on, you might need to account for the fact that humans making decisions based on the model may not be optimal. For example, in one project, a client initially wanted a ranked list of vehicles, ordered by descending probability of component failure. After I trained the first version of the model, I learned that the client cared about certain subpopulations more than others.

7. Everyone's terminologies are different.[62] Stick to simple descriptions; avoid overloaded words and buzzwords.

With the hundreds of modeling projects I've done over the years, I think I've learned a lot. I learned how to be fast. I learned how to respond to unreasonable ML requests in record time, even if it meant hackily build a new model in 2 hours to beat some result.[63] I know this is a function of short, fast projects, which probably are more prevalent in classes and startups.

[62] For instance, when I say "algorithm," I mean a series of steps that I control. Maybe this is what other people also mean, and there is confusion because different stakeholders control different things – for example, the "algorithm" for me is how I train a model; the "algorithm" for a client is the entire end-to-end ML product that shows up on their computer in true Software-as-a-Service form.

[63] I became so good at hyperparameter tuning – a secret I learned is to focus on hyperparameters that determine how I subsample to construct the training set. One such example of a hyperparameter is how I downsample the majority class.

But this kind of work made me realize that although I absolutely love distilling a problem into an easy question and coming up with an answer as quickly as possible, I also want to spend long amounts of time dwelling on this information to think of something long term, the "right" solution.[64] Although startups might be a great place to do the former, you cannot do the latter at a small startup. When I realized this, I knew it was time for me to move on. Leaving a place that taught me so much was truly difficult.

[64] To build something that repeatedly delivers value, you can't escape the responsibility of good, methodical science.

Parting thoughts

Learnings from one data science project can be highly applicable to the next. I would like to think that I have come a long way since my Block Dude days. I'm very proud of the last deep learning class project I did at Stanford – something personal, something impactful. I trained an autoencoder on my text message data to identify manic and depressive episodes,[65] and I didn't expect to be able to recall any episodes. But I was able to recall over 40% of episodes! It seems like a low number, but to me, it was magical, since the true baseline – myself – had a recall rate of 0%. I didn't care what grade I got;[66] all that mattered to me was that I had finally built something useful for myself with machine learning.

[65] I am diagnosed with Bipolar II, and I had several weeks of episodes. I treated this as an anomaly detection problem.

[66] I think I actually got a near-perfect score on the project, even though the "high" MSE and "low" precision and recall rates were nicely displayed in the poster.

I was not on time to the poster session. I hurried to some Arrillaga building with my poster freshly produced from FedEx's massive printer, set up some battered easel, and explained my work to the handful of people that stopped by. I was honest about my results, and that sparked conversation. Maybe because I was texting faster, manic messages had more typos and thus a higher MSE, someone said. I liked that comment. I made a mental note to add the length of the message as a feature for the linear model I was planning to train on top of the autoencoder's hidden states and other simple features. My data wasn't perfect, and I could use the imperfections to my advantage instead of try to clean the data further.

The poster session ended, and I biked to work for a client meeting. The diagnostic trouble codes that occur in a vehicle couldn't be used to accurately predict failure, someone said, because many vehicles had tens or even hundreds of codes fire consecutively – sometimes for no good reason. Maybe this was because of a networking problem. Or a faulty sensor.

"What do we do about it? Should we drop duplicate consecutive codes?" they suggested.

I thought for a moment, then responded. "It could be a feature."

Acknowledgements

This has turned out to be much longer than I expected, and I feel like I have only scratched the surface of all that I have learned from this field. I feel immensely privileged to have learned from and worked with smart and well-known people and institutions. But for the first time in my life, I feel incredibly burned out,[67] and I do not have a plan for what to do tomorrow. A large part of my identity has revolved around programming and modeling, and I'm happy that some of my strong emotions and thoughts (at least about predictive modeling) have made their way out of my brain and into this document.

I want to give a special shout out to Alex Tamkin and Brad Ross. Thank you for your encouragement and feedback on this essay, and more importantly, thank you for being in my life.

[67] A small part of me wonders why I feel burned out so early in my career. I really did enjoy studying and working in machine learning. I fell in love with the idea of programming machines to learn from the world around them, and I met some of my best friends in this journey. I am lucky to be in a field where I can take time off – machines will still be there when I'm ready to return back to programming.

References

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

Donald E. Knuth. Computer programming as an art. *Commun. ACM*, 17(12):667–673, December 1974. ISSN 0001-0782. DOI: 10.1145/361604.361612. URL https://doi.org/10.1145/361604.361612.

Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks: A unified approach to action segmentation. *CoRR*, abs/1608.08242, 2016. URL http://arxiv.org/abs/1608.08242.

Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2 (1):56–67, Jan 2020. ISSN 2522-5839. DOI: 10.1038/s42256-019-0138-9. URL https://doi.org/10.1038/s42256-019-0138-9.

Chris Olah. Understanding lstms, Aug 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin.

Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Wikipedia contributors. Mean squared error — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=995577263. [Online; accessed 5-January-2021].

Wikipedia contributors. Extract, transform, load — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=Extract,_transform,_load&oldid=998138895. [Online; accessed 5-January-2021].