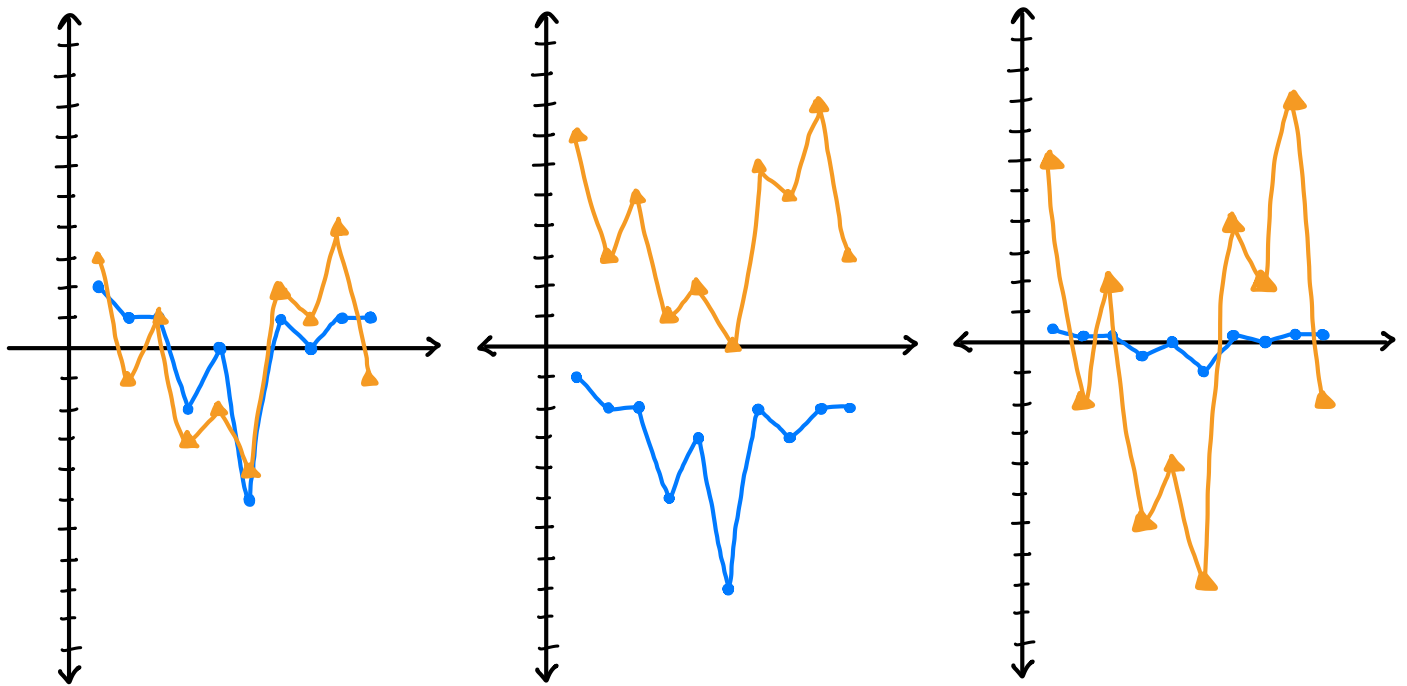


Quant Basics

A BS*-Free Guide to the Intersection of Math and Equities Trading

*Bad Science



Allison Bishop and Lisa Oakley
Proof Trading, February 2026



Quant Basics: A BS^* -Free Guide to the Intersection of Math and Equities Trading

Allison Bishop and Lisa Oakley
Proof Trading

February 2026

Contents

1	Introduction	3
2	Financial Data Basics, and why you should know them before you start	5
2.1	Important Terms	6
2.2	Financial Data Sources	6
2.3	Example: Looking at Basic Trade Data	7
2.4	Checking Our Data and Assumptions	10
2.5	Organizing Computations on Financial Data	11
2.5.1	Example: how query times vary by query structure	14
3	Anatomy of a performance metric	17
3.1	Example: Slippage Vs. Arrival	18
3.1.1	What are our Data?	19
3.1.2	Pointwise Slippage Vs. Arrival	19
3.1.3	Weighting the Pointwise Metric	20
3.2	Structuring Metrics in General	21
4	Probability Distributions: everyone's noncommittal friend	23
4.1	Describing Small, Discrete Distributions	24
4.2	Describing Large, Infinite, and Continuous Probability Distributions	27
4.3	Gaussian Distributions and the Central Limit Theorem	29
4.4	Probability Distributions in the Natural World (and Financial Markets)	31
5	Data Normalization, or how to read one story with thousands of characters	34
5.1	Adjusting by Average Daily Volume (ADV)	35
5.2	Adjusting to Relative Price Movements	36
5.3	Grouping	36
5.4	Re-Centering	37

6	Distribution Testing	40
6.1	Uniformity Testing	44
7	Hypothesis Testing, Monte Carlo Simulations, and when it's ok to skip the math	48
7.1	Pitfalls and Replication	50
7.2	Monte Carlo Simulations	51
7.3	Random Walks, and the Downside of Acting on Questionable Hypothesis Testing Results	54
8	Correlation Testing	59
8.1	Numerical Variables	59
8.1.1	Investigating the Pearson Coefficient	63
8.1.2	Investigating the Spearman Coefficient	65
8.2	Categorical Variables	66
8.2.1	Investigating the Chi-Squared Test	68
8.3	Mixing Numerical and Categorical Variables	71
9	An Overview of Machine Learning and Optimization Problems	72
9.1	Supervised Learning	73
9.1.1	Offline Supervised Learning	75
9.1.2	Online supervised learning	75
9.2	Reinforcement Learning	83
9.3	Unsupervised Learning	89
9.3.1	Clustering	91
9.3.2	k-Means Clustering	92
9.4	Machine Learning and Optimization	96
10	Model Types and Fitting Models	97
10.1	Lookup Tables	100
10.2	Linear and Polynomial Models	104
10.3	Logistic Models	107
10.4	Splines	110
10.5	Decision Trees	112
10.6	Neural Nets and Deep Learning	119
10.6.1	Gradient Descent and Sigmoid Functions	123
11	Closing thoughts	128

Chapter 1

Introduction

These days, there is a proliferation of data science boot camps, blogs on machine learning, and widely available software tools for getting started with data analyses. But the topic of quantitative research on financial data (particularly trading data) remains quite difficult to learn. Some reasons for this are historical: the US equities market, for example, has evolved as a complex system that was not designed to a single, unifying vision. Artifacts of its structure show up as idiosyncracies and edge cases in data that you have to be “in-the-know” to understand. Some reasons are cultural: a common attitude in finance is that if you try to model something and it works, you quietly use it to make money. If it doesn’t work, that’s when you write a whitepaper. Some reasons are economic: getting access to high quality granular data at scale can be prohibitively expensive.

There has never really been a strong incentive to provide good learning materials for practical quantitative research on financial data. In fact, most of the people who would be in a position to create such materials are actively disincentivized by more urgent demands on their time and their employers’ intellectual property claims. So when Proof hires scientists to do research on financial data, we face a familiar dilemma: how can we set them up to be successful quantitative researchers? An easy answer could be: hire only people who are already experienced in quantitative research on financial data. We reject that answer for a few different reasons. For one, it’s a closed loop that leads to entrenched ideas and entrenched demographics. But also - it’s just not very Proof-y. For a tiny company, we have a strong combination of expertise in market structure, trading dynamics, machine learning, and science communication. This puts us in a good position to hire and train strong scientists who start without specific finance experience, and we believe that doing so will make our research more innovative. This is why we’ve opted to take the path of generating our own learning materials to train quantitative researchers.

It is our intention that these materials be accessible to a broad audience of people with

a quantitative background, and so we try not to assume familiarity with too many specific statistical tools or concepts. In places where we do draw heavily on external concepts, we make an effort to point out supplemental resources that might be useful to fill in any background gaps. We do not make an effort here to cite all sources that might be relevant, but rather stick to citing the ones that we personally have drawn upon in formulating our own understanding, as well as those that are historically foundational.

We try to break out the material into short sections, many of which can be read on their own. For readers who may be beginning with little to no exposure to US equities trading, we recommend starting with Proof's market structure primer as necessary background. For readers with strong computer science backgrounds (particularly in machine learning), the later sections may be rather rudimentary for you, and you may get the most out of the early sections which center more on domain-specific topics. For readers with strong equities backgrounds, you may find the early sections rather tedious, and may get the most out of later sections which use trading examples to teach some basic machine learning concepts.

Chapter 2

Financial Data Basics, and why you should know them before you start

Programming languages like Python (and the myriad packages that sit on top of it for performing data analyses) are often designed to be agnostic to the nature of the underlying data. This is a useful feature - it's nice that we don't have to learn a new programming language every time we sit down to analyze a new kind of data. But this can also lead to a false sense that you don't deeply need to know the nature of your data and how it is organized.

This is going to sound obvious, but the first thing one should always do as a data analyst is *look at your data*. Does it make sense? Is it roughly what you expected? Are there the right amount of data points? Do you see all the columns you will need, and do you know what the values/units in them mean? If not, don't race ahead and figure it out later. This is too likely to lead to errors down the road. "Oh, you needed that report in shares and these are dollars? oops," or "hmm... how come my complicated code to compute projected prices is projecting a price of -100 dollars per share?"

If there's some basic property of your data that you're not sure about, what do you do? Well, there are three general strategies: (1) consult documentation, (2) ask someone, (3) cross reference with another data source. We roughly recommend doing them in this order - even if you usually end up asking someone because the documentation is missing/confusing/incomplete/outdated, etc., it's often good to check this first so that when the person you ask barks at you "check the documentation!", you can say, "I did, and it doesn't say what code X means." There are often times where (3) is a good strategy. For instance, if we're looking at data for trades of Microsoft stock and we see a value like 22851 for price, we might wonder if this represents dollars, cents, or some other decimal unit of precision. We can check Yahoo! Finance and see that typical prices for Microsoft stock are closer

to hundreds of dollars these days, rather than thousands of dollars or tens of dollars etc., making it clear that our value is delineated in cents and the corresponding value in dollars is \$228.51.

2.1 Important Terms

We will discuss many of these topics in more detail, but here is a glossary of important terms that we will use throughout the guide, and what we mean by them.

Exchange/Trading Venue	Corporate entities which provide a platform for buying and selling securities and have been granted exchange licenses by the SEC, giving them a particular set of obligations and benefits. NYSE, NASDAQ, and IEX are examples of exchanges.
Dark Pool	Corporate entities which provide a platform for buying and selling securities and have been granted licenses as Alternative Trading Systems (ATSs) by the SEC, giving them a different set of obligations benefits. The name “dark pool” comes from the fact that ATSs tend to differ from exchanges by not disseminating public price quotes, but this distinction has been blurring in recent years.
Equity/Equity Security	Stock or similar (e.g. an exchange-traded fund/ETF) traded on exchanges.
Symbol	Literally refers to the name of a specific equity (e.g. BAC for Bank of America), but is commonly used to refer to the equity itself.
Bid	The price at which someone is offering to <i>buy</i> some quantity of an equity
Ask/Offer	The price at which someone is offering to <i>sell</i> some quantity of an equity

2.2 Financial Data Sources

There are many different data relating to historical market activity that are accessible through a variety of sources. The data source that we use throughout this guide is TAQ (Trade and Quote) data, which is disseminated from the Securities and Information Processor (SIP). “Trade and Quote” refers to the contents of the data. These data contains information about all trades made on the exchanges, as well as National Best Bid and Offer (NBBO) quotes of what quantities and prices equities are available for purchase or sale on the exchanges. The National Best Bid (NBB) represents the highest quoted price to buy across the exchanges, and data includes the total aggregated size (number of shares) at this

price. The National Best Offer (NBO) analogously represents the lowest quoted price to sell. The NBBO refers to the combination of the NBB and the NBO. We can visualize these two kinds of data as in Figure 2.1.

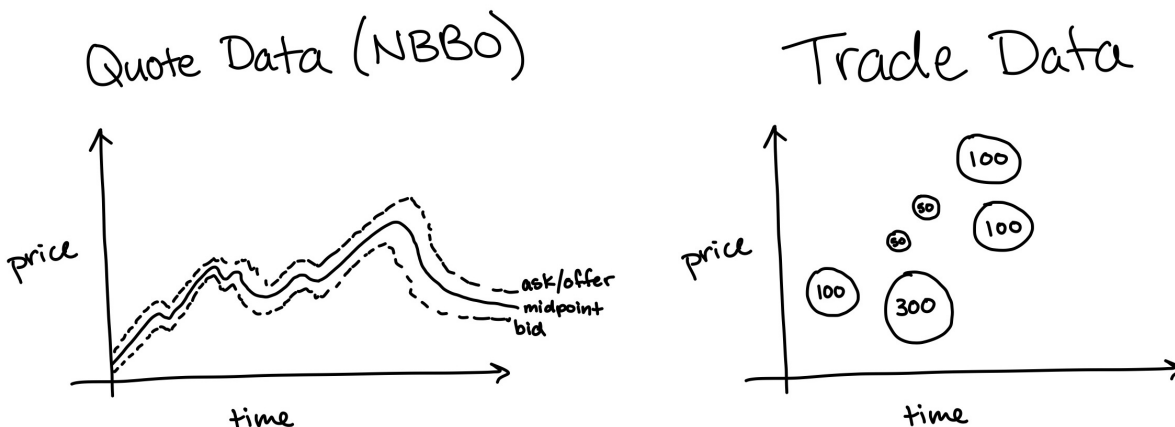


Figure 2.1: Two graphical representations of TAQ data for an arbitrary symbol. The left hand example shows the typical representation of the “spread,” or difference between the bid and ask prices in the NBBO quote with the middle line representing the midpoint price at each time. The right hand example shows various executed trade prices over time. The size and value of each point represents the volume traded.

TAQ data is organized by time and symbol, meaning that for each symbol, we have a time series of trades, quoted bids, and quoted offers dating back to the first trading day for that symbol. Each exchange streams its trading and top of book quote data each day to the Securities Information Processor (SIP), which compiles it into a single time series. At Proof, we access historical TAQ data through a provider OneTick, using pre-built querying tools in their cloud environment.

Other examples of data sources that a quantitative researcher may use include internal client data and data related to external market forces, such as earnings reports schedules for different companies. Regardless of data source, the overarching approach to financial data analysis discussed in this guide is still applicable. In some cases, other types of data might have additional concerns, such as privacy of client information.

2.3 Example: Looking at Basic Trade Data

Let’s take a look at some basic data for trades of Bank of America stock on September 1, 2022 (these examples are chosen arbitrarily).

```
In [10]: 1 T = LookAtTrades(dt.datetime(2022,9,1,9,31),dt.datetime(2022,9,1,9,32),['BAC'])
          2 T.head()

Out[10]:
```

		Time	PRICE	SIZE	COND	PARTICIPANT_TIME	EXCHANGE	SYMBOL_NAME	TICK_TYPE
0	2022-09-01 09:31:00.225022	33.240	78	I	2022-09-01 09:31:00.224774000	J	NYSE_TAQ::BAC	TRD	
1	2022-09-01 09:31:00.231072	33.245	98	I	2022-09-01 09:31:00.230361871	D	NYSE_TAQ::BAC	TRD	
2	2022-09-01 09:31:00.361722	33.240	44	F I	2022-09-01 09:31:00.361374840	T	NYSE_TAQ::BAC	TRD	
3	2022-09-01 09:31:00.361727	33.240	100	F	2022-09-01 09:31:00.361374840	T	NYSE_TAQ::BAC	TRD	
4	2022-09-01 09:31:00.361822	33.240	1	F I	2022-09-01 09:31:00.361524036	U	NYSE_TAQ::BAC	TRD	

The function “LookAtTrades” here that is pulling the data from its original source into a locally running Jupyter notebook. The details of LookAtTrades are specific to our data provider OneTick and their query language, so it would not be particularly insightful to share them here. An important thing to know here is that OneTick is fundamentally sorted by time, meaning that the first 5 records we are viewing here represent the first five trades with timestamps starting from 9:31 am on September 1, 2022.

A few properties of our data are illustrated by this sample. (These are properties we know from context - they are certainly not all things you could be expected to magically figure out from the data alone!) Each row represents a trade, the prices are in dollars, and the sizes correspond to the number of shares. The SYMBOL_NAME column here contains strings that indicate which security is being traded. The prefix NYSE_TAQ indicates the data source (in the way that OneTick indexed data at that time), while the ticker “BAC” is the shorthand code for Bank of America stock. The TICK_TYPE column is merely telling us that these are trade records, rather than quotes or some of other kind of financial data record. The COND column contains condition codes that give us additional information about what kind of trade this is. The meanings of these codes can be found in the publicly available Daily TAQ Spec. The I, for instance, indicates an “odd lot” trade, which means that the size of the trade is smaller than the “round lot” unit of 100 shares. This is why the I designation is missing from the trade of 100 shares that appears fourth in this tiny sample. [Historical note: in 2022, most symbols had a round lot of 100 shares, but more flexible round lot sizing has been introduced recently as of this writing.] The F code indicates an “Intermarket Sweep Order.” We are not going to go into the details of what all such things mean here, because it would be a long detour from our purposes, and is not important for understanding the examples we will work through here. One important thing to note about the COND column here is the fact that the codes are not mutually exclusive, and a single trade can have multiple codes. The EXCHANGE column, however, always has a single letter that indicates the exchange where the trade occurred. Off-exchange trades are grouped together under a single letter. The mapping between exchanges and letter codes here can similarly be found in the Daily TAQ Spec.

Let's examine the two time columns for a moment. One thing we might notice is the differing precision: the PARTICIPANT_TIME column here is presented with nanoseconds, while the Time column is presented with microseconds. Time is a crucial quantity when it comes to understanding financial data, so it is worth taking a moment to reflect on what these two different times represent. Intuitively, we often think of time as being attached to an event: “what time did X occur?” we may ask. But this ignores an important subtlety - time according to what clock? Time is only truly defined through our measurement of it, and any two different clocks can be out of sync. The PARTICIPANT_TIME column here represents the time on the exchange's clock when it matched the trade. The Time column represents the time on the clock of the Securities Information Processor (SIP) when it processed the trade report from the exchange. Figure 2.2 illustrates an example of what we mean by this.

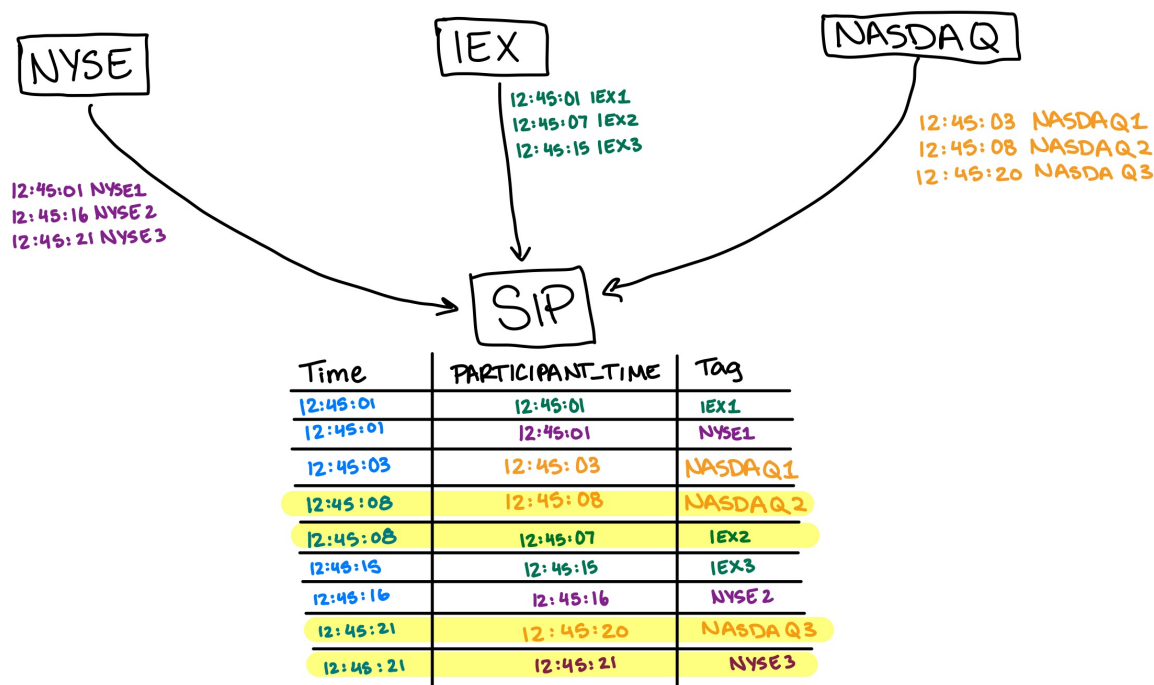


Figure 2.2: TAQ data time fields. Each exchange streams data to the SIP with internally labeled timestamps. The SIP unifies these time series with its own global time field (“Time”) based on when it was processed on the SIP, and maintains the exchange-defined time (“PARTICIPANT_TIME”) as a subfield. The highlighted rows show examples of when the SIP time might not agree with the time or ordering of trades based on the internal exchange times. Within each exchange, TAQ data in the SIP should maintain consistent order.

Since the clocks of different exchanges can be out of sync from each other slightly, using “Time” as our primary timestamp enables us to keep everything ordered relative to a single

clock. However, if we are analyzing some phenomenon of trading at the microsecond or nanosecond level, we may prefer to use the exchange timestamps. If every clock were perfectly in sync, we would expect `PARTICIPANT_TIME` to always be earlier than `Time`. For our small sample this is true, but due to clock drift, we would expect to see some minor violations of this in a larger data set.

2.4 Checking Our Data and Assumptions

Let's get a little more systematic now about examining our data, beyond just looking at a few sample data points. For each field, there is a type (e.g. number, character, string, timestamp) and a set of valid values (e.g. non-negative integers, strings with lower case letters). Numerical fields will also have associated units. Prices, for example, may be in dollars, cents, or mils, etc. Sizes might be in shares or round lots. Some fields may be optional, and may occasionally be empty or populated with defaults. Some defaults may contribute problematically to aggregate analyses if not properly handled. For example, if you are looking at data points that represent orders, `minQty` may be marked as -1 when there is no minimum quantity. Any raw averaging over the `minQty` field with this default still included will be nonsensical, but the final result might still look reasonable on its face.

When starting to analyze a new kind of data, it is a good exercise to write down all of the things you think should be true of your data. It's best to do this starting zoomed in on individual fields in individual data points, and then gradually zooming out. For example, you could write down everything you think should be true for individual values in your data set. For our trade data example above, this would include things like: trade prices should be > 0 , and trade sizes should be positive integers. Exchange codes should be single characters contained in a particular list of prescribed values. And so on. You can then go and check for violations of each of these things, and figure out what they mean. This can catch idiosyncracies like the `minQty = -1` case that you may not have known about.

Next zoom out a bit and write down everything you think should be true for rows, columns, or other natural groupings of your data. This would include things like: percentages for comprehensive and mutually exclusive categories should sum to 1, or the column of timestamps in a sorted time series should be non-decreasing. Perhaps there are relationships that should hold across columns, like that notional value is the product of price and size. Checking for violations here can alert you to upstream data quality issues, or inspire additional pre-processing.

Finally, zoom out and write down any properties that you think should hold globally or over larger portions for your data. For example, is there an identifier field that should be

globally unique? Or unique within a particular trading day? Checking these things now can help sort out misunderstandings you may have had about your data specifications that could really bite you later on.

You can repeat the same kind exercise with things that should be approximately true rather than precisely true. For example, is the raw number of data points roughly in line with your expectations? Do the price distributions look like you might expect? For our sample trade data above, we might ask: is `PARTICIPANT_TIME` earlier than `Time` for nearly all data points? Even if these kind of questions don't lead you to discover any issues, they can be great for building up your intuition about your data.

It is often useful to intersperse these precise and approximate checks with your computations. If you are about to divide by something, it's good to check that it isn't zero. If you are about to drop null values from your data set, it's good to check how many there are and whether that seems reasonable. If you do a complicated sequence of steps that should not change the number of data points you have, it's good to check that this invariant holds. Rushing into and through data analyses without doing these things can lead to a lot of mistakes and wasted time.

Overall though, there is no substitute for just looking at your data: both raw and after each significant processing step. Automated checks are also crucial for screening large data sets for potential discrepancies that may hide in the sheer mass of data, but it's pretty surprising how often things can be caught just by looking at the data yourself. With your eyeballs, or whatever else you use. Analyzing data without frequently looking at it is like reviewing a movie without actually watching it. Sure, you can get a sense of it from other people's summaries, but it isn't really the same thing.

2.5 Organizing Computations on Financial Data

Once we understand a bit about our data and what it represents, we will naturally be eager to do some computations on it. There are a lot of different ways we can do the same thing, and which of those ways are fast and which of those ways are untenably slow can depend on how our data are stored and how our data analyses tools are implemented.

Specifically for data about US equities trading, computation time really matters because the data itself is huge. As of this writing, there are roughly 1200 symbols trading each day. [Reminder: we use "symbol" as an umbrella term that encompasses stocks as well as objects that trade like stocks on exchanges, like exchange traded funds (ETFs).] There is a high level of variation in the amount of data per symbol. If we are analyzing trades, a very active symbol like SPY (an ETF that tracks the S&P 500) may have something like a million trades

in a day, while a very inactive symbol might have hundreds or even less. The number of quotes might be an order of magnitude or two greater than the number of trades. If you are doing a computation that involves many symbols over many trading days, the total size of the relevant data can quickly become overwhelming. At this point, if you think of your data as an unorganized blob and interact with it generically, you are likely to get unacceptably poor performance in your computations. We can do much better if we know how our data is already stored and organized and use that to design tailored computations that can complete quickly.

One way to think about US equities trading data is that it is a union of about 1200 individual time series, one for the trading activity in each symbol extending back to its first trading day. We could potentially store it this way - maintaining 1200 individual data sets, extending each one with new data at the end of each trading day as it happens. If we do this, pulling a subset of data that corresponds to one or a few symbols should be quite fast: reading segments of memory that are stored close together is generally faster than reading segments of memory that are stored far apart. However, pulling a subset of data that corresponds to many symbols over a single day may be slow, as we have to pull data from over 1200 potentially far apart memory locations.

A more typical organization is to store trading data in units of days, and within each day, we have the 1200 individual time series corresponding to each symbol. This means that data for SPY trading on June 17, for example, will be stored nearer to data for BAC trading on June 17 than to data for SPY trading on June 16. If we put ourselves in the shoes of people who have the job of maintaining large databases of trading data, this makes a lot of sense. Each day you have new trading data for all the symbols that you have to write to your database. It is much simpler to put this all together in one place than to split it out into separate writes for over 1200 different locations. This changes what kind of data pulls will be fast and slow. Now, pulling a subset of data that corresponds to many symbols over a single day will be fast, while pulling a subset of data for a single symbol over many days will be slower. Figure 2.3 illustrates the two data storage methods and which queries are faster in each.

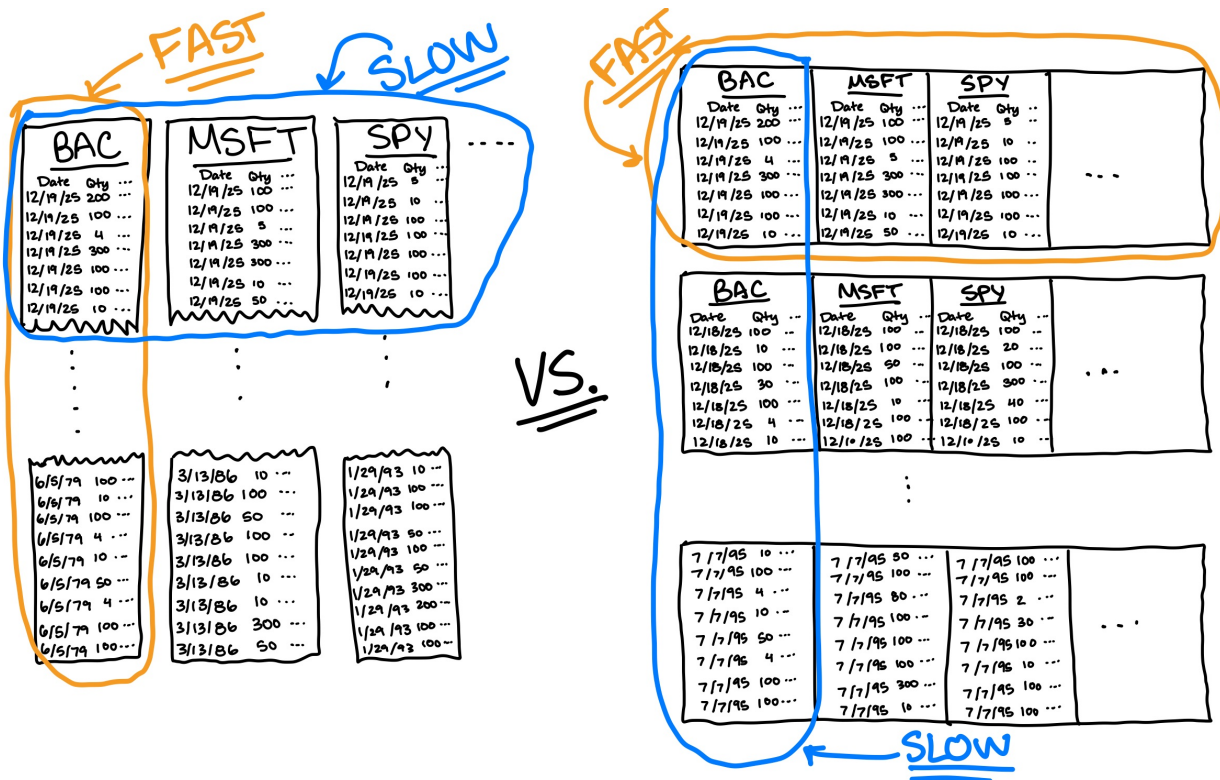


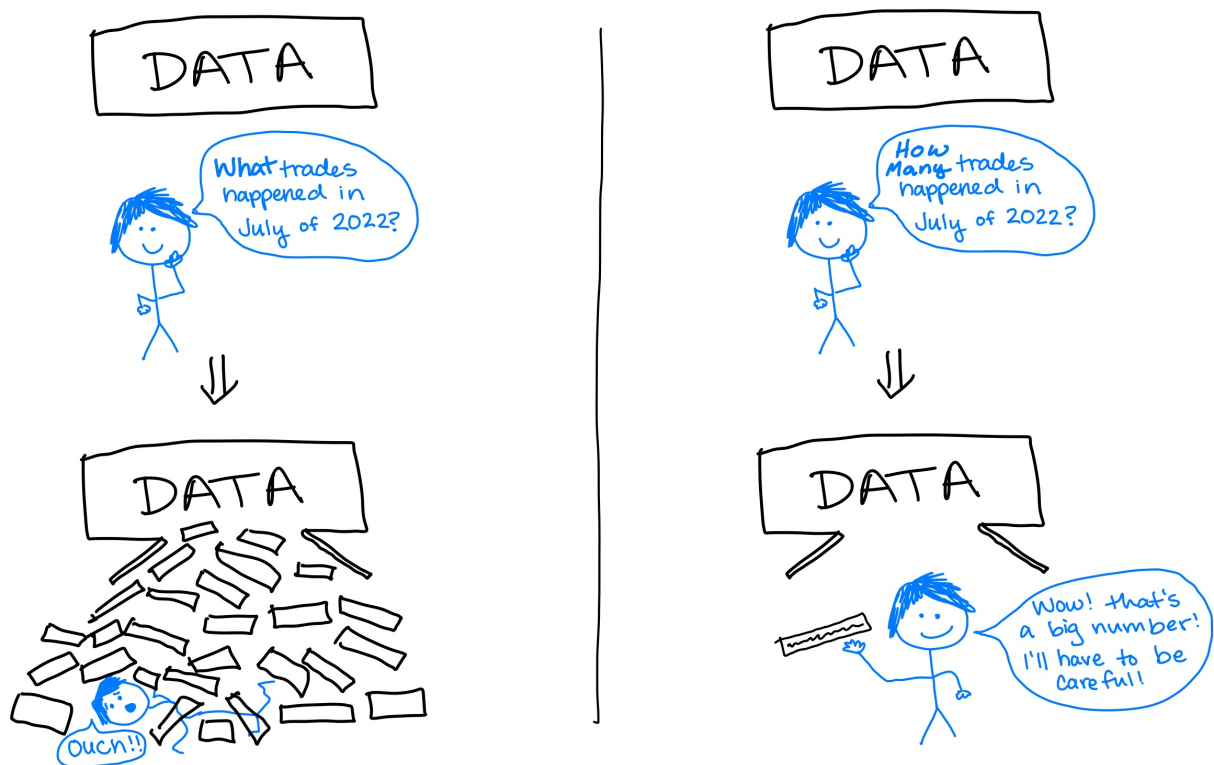
Figure 2.3: Two different methods of storing trading data. The left side represents storing data as 1200 different time series (one for each symbol) dating back to the first trading day. In this case, it is faster to pull the whole history of one symbol than one day of each symbol. The right side represents storing data as individual days with all trading for each symbol per day. In this case, it is faster to pull all data for one day than the entire history of one symbol. The right side is the more common method for storing trading data, and we assume going forward that this is how our data is organized.

Let's suppose we know that this is how our data is organized: each day of trading data is stored together, and within each day, the data for each symbol is stored together as a sorted time series. If we want to do an analysis over multiple days and multiple symbols, we should probably design our computation to analyze all of the data it needs for a given day at once, rather than, say, looping over symbols and analyzing all of the data for each symbol before moving onto the next symbol. This all becomes quite natural if you visualize data storage like a grocery store. You have a bunch of items on your list, and it is must faster to retrieve all of the items you need from each aisle as you go down it, rather than running around the store multiple times to retrieve items in alphabetical order.

It is also best to avoid moving data around if you don't need to. Pulling a bunch of data from a remote machine onto your local machine and analyzing it there may take much longer than doing the analysis in place on the remote machine and transmitting only the results.

It is also helpful to keep the data in time-sorted order, and to use analysis tools that are able to take advantage of that fact. Finding a particular data point or time range of data points, and joining data points based on timing are common operations that can be much faster when performed on data that is guaranteed to be sorted.

This may go somewhat without saying, but it's generally good to estimate how big your data is and how long something should take to compute before you set it running. If you query a database with a question like "Give me all of the trades in the market on Day X that satisfy Criterion Y," it's nice to have a rough idea beforehand how many trades this might be. Otherwise, you might end up tying up your computational resources for longer than you expected, or overloading your memory with a huge result. Sometimes it might be hard to get a decent estimate before you actually ask the query, but often it's pretty easy. One heuristic might be: how many trades in Day X *for symbol Z* satisfy Criterion Y? This will have a much smaller answer and so is safer to just run this more modest query and see the answer. You might then extrapolate by knowing what percentage of trades take place for symbol Z vs. the rest of the market.



2.5.1 Example: how query times vary by query structure

Let's see how this all plays out for a particular example. The specifics here will vary based on what database structure and software is being used, what machine the query is being

run on, etc., but we can nonetheless infer some qualitative trends. We'll consider a basic calculation of volume during regular trading hours: we total up the sizes of all market hours' trades in our database for a given symbol in a given time period. Here's what we get if we do this calculation for Bank of America stock on September 1, 2022:

```
In [29]: 1 GetVolume(dt.datetime(2022,9,1,9,30),dt.datetime(2022,9,1,16,0),['BAC'])
```

```
Out[29]:
```

	Time	SYMBOL_NAME	VOLUME
0	2022-09-01 16:00:00	NYSE_TAQ::BAC	29392819.0

The details of our function `GetVolume` here would not be insightful to show, as they are again particular to the inner workings of our database environment provider, `OneTick`. Because of the way `OneTick` is built, it is natural to build our function to take in a list of symbols and output a list of volumes, one for each symbol:

```
In [30]: 1 GetVolume(dt.datetime(2022,9,1,9,30),dt.datetime(2022,9,1,16,0),['A','BAC','TSLA','MSFT'])
```

```
Out[30]:
```

	Time	SYMBOL_NAME	VOLUME
0	2022-09-01 16:00:00	NYSE_TAQ::A	1111480.0
1	2022-09-01 16:00:00	NYSE_TAQ::BAC	29392819.0
2	2022-09-01 16:00:00	NYSE_TAQ::TSLA	51345156.0
3	2022-09-01 16:00:00	NYSE_TAQ::MSFT	19387755.0

The extent to which this query is parallelized across the symbols is controlled by `OneTick`, and the query is ultimately executing in `OneTick`'s cloud environment. The jupyter notebook here that initiated the queries and displays the results is running on a laptop. This setup will naturally affect the specific computation times we experience here, but the general trends will hold true across other typical setups.

We can run this same `GetVolume` query for *all* of the symbols in the market over a few days and measure how long it takes to get the results. If we do this over the two trading days of September 1 and 2 in 2022, this takes about 46 seconds. Notably here, we are computing over the data for all symbols for each day before moving onto the next day. Here's how we are calling the `GetVolume` query for each day inside a loop over days and timing the results:

```

In [11]: 1 #Loop over days for all symbols
          2 t_start = time.perf_counter()
          3
          4 day_start = dt.datetime(2022,9,1,9,30)
          5 day_end = dt.datetime(2022,9,1,16,0) #this should be 6 pm
          6
          7 end_time = dt.datetime(2022,9,3,0,0)
          8
          9 while(day_start < end_time):
          10     symb_list = all_symbols(day_start,day_end)
          11     if(len(symb_list)>0):
          12         V = GetVolume(day_start,day_end,symb_list)
          13
          14     day_start = day_start + dt.timedelta(days=1)
          15     day_end = day_end + dt.timedelta(days=1)
          16
          17 t_end = time.perf_counter()

In [12]: 1 t_end-t_start
Out[12]: 46.32886930000677

```

It's a little silly to set up this loop to just compute over two days, but this is a useful structure to know so it is worth spelling out. Let's contrast this with what happens we do the same computation but loop over symbols instead. This version of the query is working with the full two days of data for each symbol before moving onto the next symbol, with no parallelization across symbols:

```

In [27]: 1 #Loop over symbols
          2
          3
          4 day_start = dt.datetime(2022,9,1,9,30)
          5 end_time = dt.datetime(2022,9,3,0,0)
          6
          7
          8 symb_list = all_symbols(day_start,end_time)
          9
          10 t_start = time.perf_counter()
          11
          12 for s in symb_list:
          13     V = GetVolume(day_start,end_time,[s])
          14
          15 t_end = time.perf_counter()
          16

In [28]: 1 t_end-t_start
Out[28]: 5866.4486129

```

This time it took about 5,866 seconds, which is more than 126 times as long as the first way. It is likely that OneTick is parallelizing over symbols to some extent in the backend for the first case, but it is unlikely they are devoting at least 126 times as much computation power. Rather, this stark difference illustrates how important it is to align the structure of your queries with the way the data is stored.

Chapter 3

Anatomy of a performance metric

Now that we've established what our data is and how it is organized, it's time to think about what we want to do with it. One of the most common things people want to do with trading data is measure trading performance, so we will spend some time now digging into what this means. It's pretty common in the financial industry to talk about performance metrics for trading algorithms in shorthand. People will say: "what was the slippage vs. VWAP?", not "what was the result when you subtracted your average trade price from the volume-weighted average price of trading across the market for the interval of trading that began when your order was submitted and ended when your order was completed, and then divided by the NBBO midpoint price as of the time your order arrived?" It's not hard to see why people don't say the latter. We almost fell asleep just writing it.

But unfortunately, fully defining a metric takes that much fine print. When we collapse a large and complicated set of data to a number, we make a lot of choices that affect the outcome in large and small ways, and most of these choices are not widely standardized. This is typically the part of the data analyses where we abandon the richer landscape of distributions and "summarize" or compare them with averages or other supposedly representative stats.

Let's think through how this works for the case of a metric that is meant to measure the performance of a trading algorithm. One decision we need to make is: what base units are we measuring? Perhaps we are interested in absolute trade prices in dollars, for example. Another decision we need to make is: what is our benchmark? It may be obvious that it is good to buy low and sell high, but compared to what? A popular choice here might be to compare to the "arrival price," meaning the price of the symbol at the time the order "arrives." Though on the surface this is a straightforward metric, even this simple phrase encodes its own set of little decisions. Like, when does an order "arrive?" Is it when the sending party sends it, or when the receiving party receives it, or when it first trades, etc.?

Also what exactly do we mean by “price” at any given moment in time? The prevailing NBBO midpoint? The prevailing NBB or NBO (depending on if the order is to buy or to sell)? The price of the most recently completed trade in the same symbol, etc.?



Once we’ve decided what we’re measuring and what we’re comparing it to, we should also decide the nature of the comparison. Are we going to subtract the order’s average trade price from the benchmark price if we are buying? Are we going to divide the result by the benchmark price to get a relative result instead of an absolute one?

3.1 Example: Slippage Vs. Arrival

To illustrate this process concretely, let’s work with a laughably small fake data set of orders and look at their slippage vs. arrival. [Note: slippage vs. arrival price is a commonly used metric that grades performance by seeing how much your price “slips” with respect to the price at the time the order “arrived.” These terms can be defined many ways, but the basic idea is that a “good” price is not worse than the price when the order is placed.] The following five rows are fake orders, each corresponding to a set of fake trades that occurred within a single trading day. We list the side (buy or sell), the total size traded (number of shares), the average price per share, the total notational value of the trade (avg price times total size), the time we received the order, the time of the first fill (aka trade), the time of the last fill, and the NBBO midpoint as of the time of the first fill.

Side	Size	AvgPx	NV	TimeReceived	TimeFirstFill	TimeLastFill	ArrivalNBBO
Buy	20000	\$ 100.234	\$2004680	09:20:43.452	09:30:01.023	16:00:02.425	\$99.98
Sell	1246	\$26.04	\$ 32445.84	10:04:34.002	10:06:53.341	11:23:43.459	\$26.02
Sell	156000	\$62.935	\$ 9817860	09:45:23.224	09:45:23.654	15:59:59.023	\$63.01
Buy	5000	\$10.91	\$ 54550	14:23:52.025	14:24:01.344	14:58:43.674	\$11.33
Sell	7290	\$245.842	\$ 1792188.18	11:34:34.974	11:39:36.005	16:00:01.352	\$246.025

3.1.1 What are our Data?

In selecting the fields for this table, we’ve already made some notable choices. For one, we’ve chosen to use the NBBO at the time of the first fill as our arrival benchmark, instead of the time the order was originally received. One reason to do that is that orders may be received pre-open (like the first order here that arrived before 9:30 am) that are intended to start trading only on and after the opening auction. Pegging these to a pre-open arrival price therefore makes less sense. Naturally, we could still use “TimeReceived” as the time to set our benchmark for intraday orders (those arriving between 9:30 am and 4 pm), but here we are making the choice to just use the time of the first fill, which is reasonable in some contexts though this may blind us to impact on the price that we could have had even without trading (e.g. by posting orders that didn’t fill). We’ve also made a choice to display our timestamps with millisecond level precision. If we were looking at short-term phenomena for individual fills (e.g. 1 second markouts), we’d probably want greater precision (e.g. microseconds or nanoseconds). But for looking at overall performance over the lifetime of a order lasting for several minutes, hours, or day, millisecond precision is probably more than sufficient.

3.1.2 Pointwise Slippage Vs. Arrival

Next let’s compare our average price to the arrival benchmark we’ve chosen. For a buy order, like the first one in the table, we’ll take our average price and subtract the benchmark price: $100.234 - 99.98 = 0.254$. Currently, the units of this value are in cents per share, and a positive value represents that we paid *more* than the benchmark price. (It doesn’t matter which way you choose to do the signs, only that you keep track of what you choose.)

To get our slippage vs. arrival metric for this point, let’s divide by the arrival price: $\frac{0.254}{99.98} = 0.00254\dots$, and then divide by .0001 to get about 25 bps. [Note: “basis points” or bps (pronounced: “bips”) is a commonly used unit that refers to 1/100th of a percent.] We can perform the analogous calculation for each of our orders (multiplying by -1 for sell orders to reflect that buying low is analogous to selling high, etc.):

Side	Size	AvgPx	NV	...	ArrivalNBBO	SlippageVsArrival
Buy	20000	\$ 100.234	\$ 2004680	...	\$99.98	25 bps
Sell	1246	\$ 26.04	\$ 32445.84	...	\$26.02	−8 bps
Sell	156000	\$ 62.935	\$ 9817860	...	\$63.01	12 bps
Buy	5000	\$10.91	\$ 54550	...	\$11.33	−371 bps
Sell	7290	\$245.842	\$ 1792188.18	...	\$246.025	7 bps

3.1.3 Weighting the Pointwise Metric

After we make all of these decisions, we’ve decided on a way to grade a single order relative to a single benchmark. But probably we’re going to want to aggregate over many orders to get a sense of overall performance. So we then have to decide: which orders do we want to aggregate over? All of the orders within a certain time period? Do we need to filter out any, like any that have missing data or some that represent a different trading objective we aren’t focusing on right now? Should we do anything special to deal with outliers? And finally, how are we going to weight our aggregation? Do we want to weight individual orders by notional value or by volume, or something else?

In our toy example here, we might take a weighted average over our 5 orders, weighting by notional value:

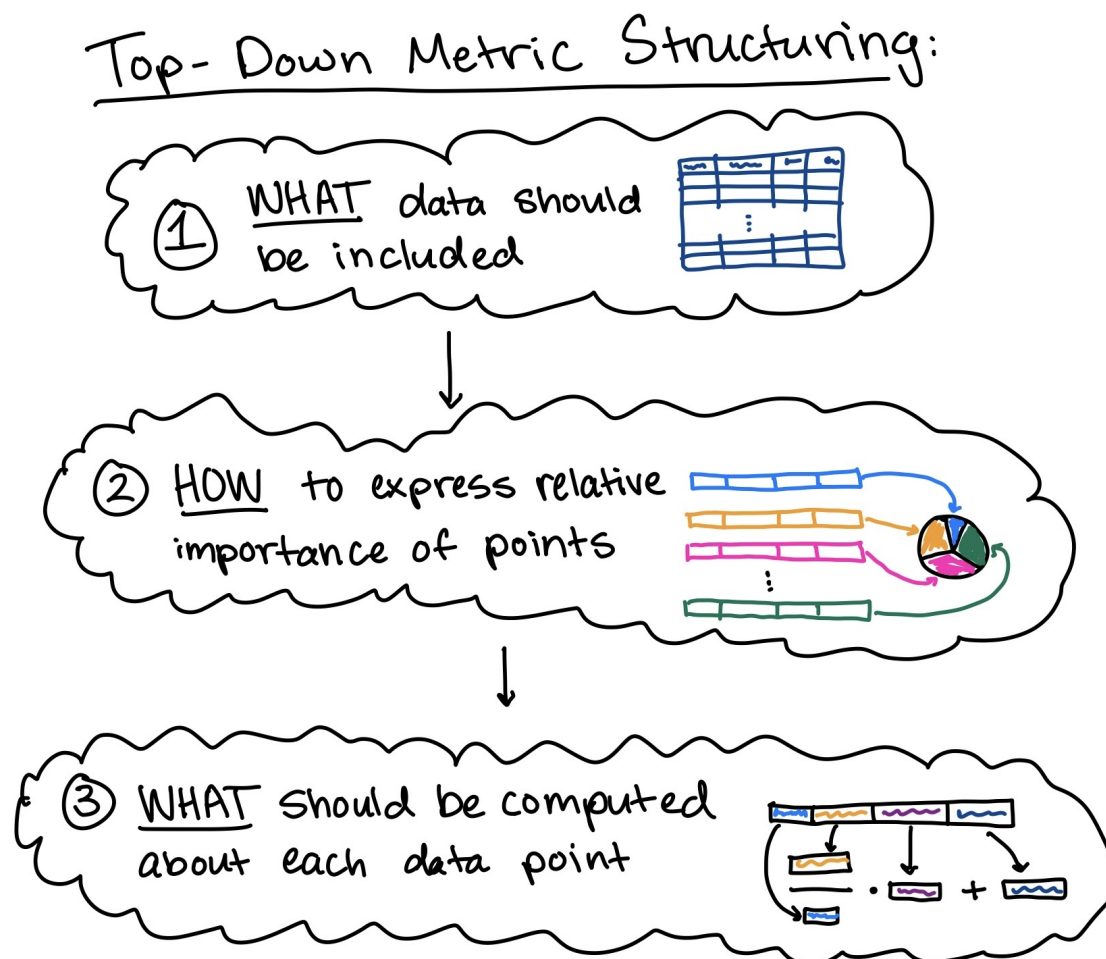
$$\frac{25 * 2004680 - 8 * 32445.84 + 12 * 9817860 - 371 * 54550 + 7 * 1792188.18}{2004680 + 32445.84 + 9817860 + 54550 + 1792188.18} \approx 11.68bps$$

Since we are staring at this in detail, we can see here that the one relative outlier (the −371 bps), is in a sort of tug-of-war with the heavier weighted value of 25 bps. This makes the overall average less of a description of an “average” order and more of a balance point between competing outliers. If our data is much bigger, this still might be what is fundamentally happening, but it may be much harder to see. One way to gauge the robustness of our average value is to recalculate it with the highest values or lowest values dropped, and see how much it changes. Here, if we throw out our lowest value of −371, we get a weighted average of about 13.21 bps instead. If we throw out our highest value of 25 bps, we get a weighted average of about 9.39 bps instead.

Since we are dealing with a weighted average here, it’s not always the case that the highest or lowest values will have the greatest effect on the average. There may be some more mild values with higher weight that are ultimately exerting more of an influence on the answer. Generally, it is a good robustness check to see how much you can ultimately move the answer by removing *any* small subset of the data points. Greater development of this idea can be found in [8]

3.2 Structuring Metrics in General

Putting all of this together, the structure of a metric like this might be more intuitive if we view it abstractly and top-down instead of specifically and bottom-up. To compute a number that summarizes a data set in some way, we decide: 1. what data to include, 2. how to express the relative importance of each data point, and 3. what about each data point we want to compute. This is not a purely general template, as not every metric breaks down this way, but it's a common one.



Shorthands for templates like this one work well to communicate metrics when there are well-known default choices. This is close to true in equities trading, but not quite. There are sometimes a few common choices for something, but not always one obvious default. This means that metrics often go under-specified in conversations, and even in written analyses. There is also a potential to cherry-pick results among the variations of reasonable parameters (whether intentionally or accidentally).

To avoid these pitfalls, we recommend:

- Make sure every choice you make in defining a metric is documented thoroughly in your reports. Even if you think it is obvious.
- Make choices as consistently as you can. Don't run one metric over two months of data and another over three months, unless there is a good reason.
- Know the intention behind every step of your computation. Why are you weighting over orders by notional value vs. volume? What would it mean if it you did it the other way? Why are you computing an average as opposed to a median and vice versa? The answers may sometimes be, "well, I gotta do something," but it's still good to keep these questions front of mind.

Chapter 4

Probability Distributions: everyone's noncommittal friend

Now that we have discussed how to define metrics, it is tempting to go running about computing them and drawing conclusions. But there is a downside to rushing too quickly from theory to numbers. Knowing how to produce numbers is a far cry from knowing how to interpret them.

We describing a person, one can hardly do worse than the phrase “good with numbers.” It is preferable to “beautiful but doesn’t know it,” but not by much. Probability distributions, not numbers, are the work horses of most quantitative analyses in finance, and describing a distribution by a single number is like describing Dolly Parton as “a singer.” [Aside: the woman is a multi-dimensional musical genius, and this is an inarguable fact.] Probability distributions literally contain multitudes, and when we collapse them to descriptions like “the mean is 0 and the variance is 2,” we lose important information.

On the flip side, carrying around full distributions is a pain. And the information we have about them in the first place is likely to be incomplete or somewhat inaccurate. But to the extent that we can reasonably do so, working with distributions helps us to:

- retain a nuanced view of our context
- avoid jumping to unwarranted conclusions
- stress test our models and decisions across a wide range of plausible scenarios.

So what is a probability distribution, and how do we work with one? You may already know this, but it’s worth revisiting the fundamentals.

4.1 Describing Small, Discrete Distributions

When we’re dealing with a small set of possible outcomes, like the roll of a die or the flip of a coin, a probability distribution can be described by listing each possible outcome as well as its corresponding probability. The distribution for a fair coin flip, for example, can be described by: “Heads: $\frac{1}{2}$, Tails: $\frac{1}{2}$.” Crucially, our outcomes here are exhaustive and mutually exclusive, and our probabilities are non-negative values that sum to 1. For cases like a fair coin flip where each outcome is equally likely, we call these *uniform* distributions.

The probability distribution is a concept in its own right, and is separable from any particular process for *sampling* from it. We often have a particular process of sampling in mind when we define a distribution (e.g. flipping the coin), but this does not have to be the case. Sometimes it may be easy to come up with many ways of sampling from the distribution, other times it may be hard to come up with even one. And sometimes it will be unclear whether two sampling methods actually correspond to the same distribution.

We typically use capital letters to denote random variables, which are placeholders for the outcome of a sampling process. Sticking with our fair coin example for the moment, we can let X denote the outcome of a coin flip that hasn’t happened yet. Our description above, “Heads: $\frac{1}{2}$, Tails: $\frac{1}{2}$,” describes the probability distribution associated with the random variable X .

It can be a little confusing that there are three very related but technically distinct things floating around this picture: a probability distribution (“Heads: $\frac{1}{2}$, Tails: $\frac{1}{2}$ ”), a sampling procedure (“flip the coin!”), and a random variable (“let X represent the outcome of a fair flip...”). Often, the distinctions between these won’t make too much difference, but we will try to be precise in our usage of the terms. When we use a random variable, we typically have a combination of a probability distribution and a sampling procedure in mind, and will try to specify them if they are not clear from the context.

Yet another distinct but related object enters the picture when we sample from a distribution repeatedly. Suppose for example, that we flip a fair coin 100 times, and we get 54 heads and 46 tails. In such a case, we can define the *empirical* distribution as (“Heads: 0.54, Tails: 0.46”). If we didn’t know the original underlying distribution, (“Heads: $\frac{1}{2}$, Tails: $\frac{1}{2}$ ”) in this case, the empirical distribution might represent our best guess for the underlying distribution, based on our experimental data. If we then simulated new samples by sampling from our empirical distribution, this would *not* be the same as continuing to sample from the original distribution, though in this case they are somewhat close. Figure 4.1 shows a mapping of these concepts.

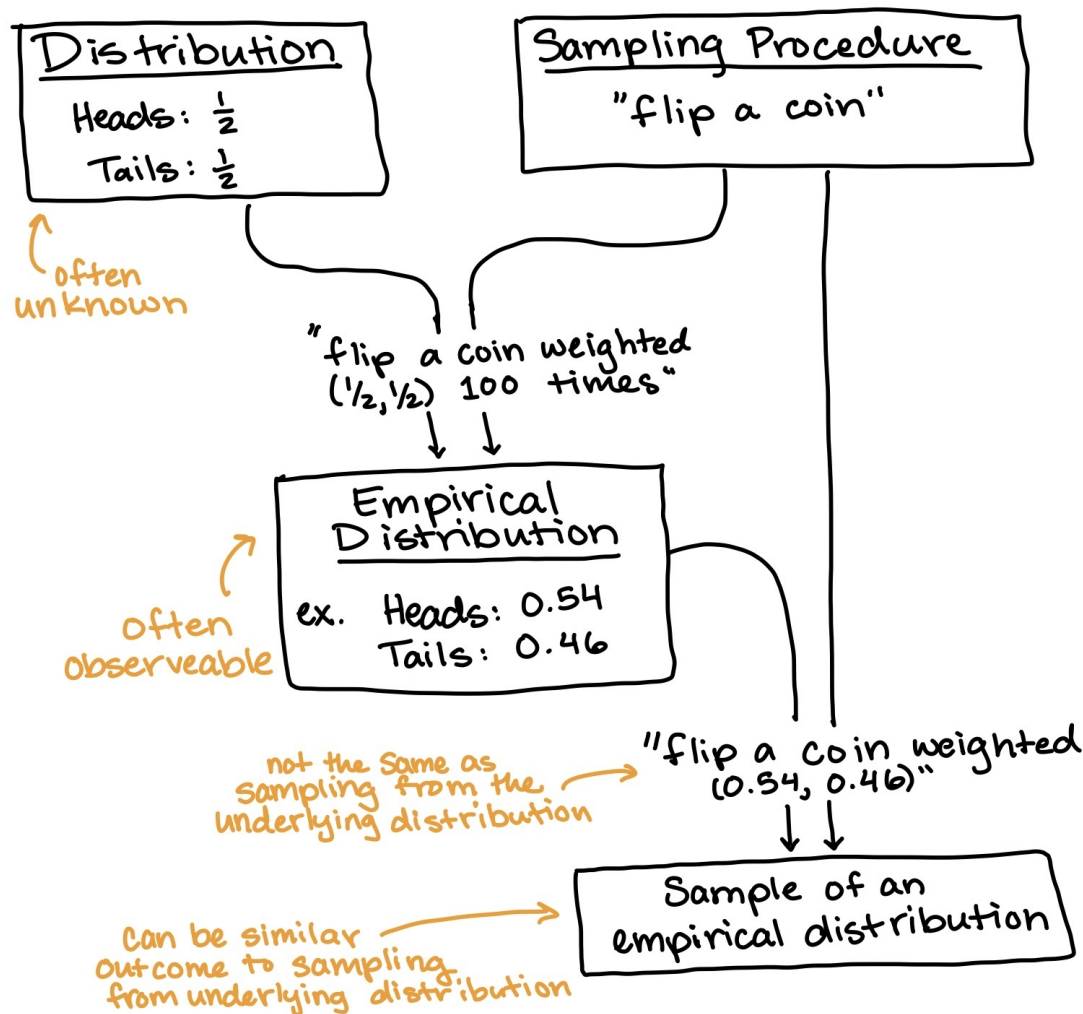


Figure 4.1: A mental map of how the distributions and sampling procedures are connected.

The *expectation* of a random variable is typically defined as:

$$\mathbb{E}[X] := \sum_{x \in S} x * p(x),$$

where S is the set of possible outcomes, and $p(x)$ is the probability of the outcome x .

Naturally, this only makes sense when the outcomes are numerical values. If we roll an evenly weighted six-sided die with outcomes 1, 2, 3, 4, 5, 6, for example, the expected value is $\frac{1}{6} * 1 + \frac{1}{6} * 2 + \dots + \frac{1}{6} * 6 = 3.5$. You'll notice something weird about this value 3.5 - it isn't one of the possible outcomes. The terminology here can lead to some confusion - the expected value is not a value we "expect" to happen. Rather it is just a numerical average, weighted by probabilities. No more, no less. On its own, it doesn't tell us anything about

how much divergence there is between common values. For example, we could get the same expected value of 3.5 for a random variable that takes on two possible outcomes, one with a value of 100 and one with a value of -100 , with probabilities 0.5175 and 0.4825 respectively.

How much the actual values tend to deviate from the expectation is captured by the *variance*, which is defined as:

$$\text{Var}[X] := \mathbb{E}[(X - \mathbb{E}[X])^2] = \sum_{x \in S} (x - \mathbb{E}[X])^2 * p(x).$$

In other words, we are taking each possible value x , calculating how far it is from the expected value, squaring that (so that being less than or greater than the expected value is treated the same), and then averaging over x 's according to their probabilities. The *standard deviation* of a random variable is the square root of the variance - this gets us back to the original units, and hence can be more directly useful sometimes when reasoning about deviation from the mean. Figure 4.2 explains how these metrics relate to the underlying probability distribution.

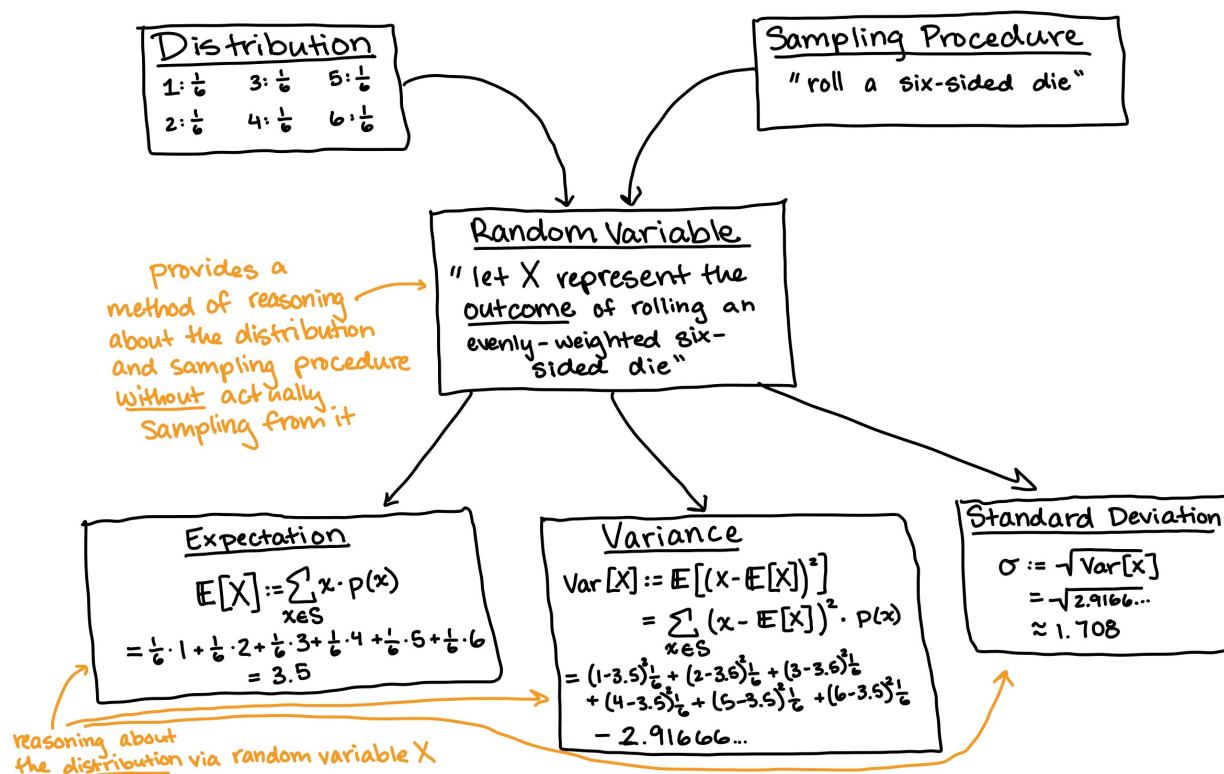


Figure 4.2: Reasoning about a distribution via random variable X .

4.2 Describing Large, Infinite, and Continuous Probability Distributions

If the set of possible values is very large (or even infinite), the approach of defining a probability distribution via a list of probability values stops being practical. If the list is just long but still finite, we don't have to adjust our thinking too much. We can specify our probability distribution by providing a function between possible values and their corresponding probabilities. For example, we can define a probability distribution on the set of positive integers from 1 to 100 by saying that for each $1 \leq x \leq 100$, $p(x) := \frac{x}{5050}$. To see that this is a valid probability distribution, note that each value is nonnegative, and

$$\sum_{x=1}^{100} x = 5050,$$

and so

$$\sum_{x=1}^{100} \frac{x}{5050} = \frac{1}{5050} \sum_{x=1}^{100} x = 1.$$

If we primarily care about the ratios between probabilities, we can take a bit of shortcut here and just say that $p(x)$ is proportional to x for each x in our set of outcomes. If we later want to compute the constant of proportionality, we can do so using the fact that the probabilities must sum to 1.

Sometimes, defining a function this way can make sense even for an infinite set of possible outcomes. For example, those familiar with limits in elementary calculus may remember how to interpret the statement:

$$\sum_{x=1}^{\infty} 2^{-x} = 1.$$

In this framing, it would make sense to define a probability distribution on the set of positive integers $1, 2, 3, \dots$ by saying that $p(x) = 2^{-x}$ for each x .

Here's a fun thought experiment: what's a sampling procedure for this distribution? How might we select a positive integer from the infinite number of possibilities, while staying true to these specified probabilities? Here's one way we could do it: we could flip a fair coin. If it comes up heads, we output 1 as our chosen sample. This has probability $\frac{1}{2}$. If it comes up tails, we keep going, and flip the coin again. If it comes up heads this time, we output 2 as our chosen sample (this has probability $\frac{1}{4}$). If it comes up tails, we keep going. And so on. In other words, we output x only if we first get a heads on the x^{th} coin flip. Is this a practical sampling procedure? Well, kind of. Most of the time, we'll finish it pretty quickly,

as it won't take too many flips to get heads. But with vanishingly small probability, we will need to keep going for a very very long time.

Things get much more confusing when the set of outcomes is not only infinite, but also continuous. For example, suppose we want to specify a probability distribution on real numbers. One that might make intuitive sense to contemplate is a uniform distribution over the real numbers between 0 and 1. If each real number in this range is “equally likely,” we can already derive some properties of this distribution, like the fact that it should place a total probability weight of $\frac{1}{2}$ on the values from 0 to $\frac{1}{2}$. But what is the probability of any particular exact value under this distribution? Essentially, it is 0! If it were *not* 0, the infinitely many values who have this probability would make the total sum of probabilities be greater than 1!

In this case, thinking of probability as a value attached to a specific outcome starts to break down. What we do instead is start to attach probability values to ranges. The probability of the range $(x, x+\Delta)$ for each $0 \leq x \leq x+\Delta \leq 1$ under this uniform distribution should be equal to Δ . This leads to sensible behavior like the total probability of the range from 0 to 1 being equal to 1.

In dealing with real numbers (or other similar settings), we can still specify probability via a function, but it helps to use an additional bit of indirection. For our uniform distribution, we can write $f(x) = 1$ as the *probability density function*, and then define the probability of any range of values R as:

$$\int_R f(x)dx.$$

In this context, our rule about probabilities summing to 1 is replaced with:

$$\int_0^1 f(x)dx = 1,$$

as our full range of possible values is between 0 and 1. Like a probability, a probability density function should also be non-negative.

We can cut a continuous probability density function into a discrete probability function if we simply bucket our values into ranges. For example, we could round our real numbers to the nearest two decimal places. The probability assigned to each rounded value would be the integral of the probability density function over the range of real numbers that round to that value.

There are many textbooks and courses devoted to the complexities of probability theory in continuous settings, which is essentially a subcase of the mathematical field known as real analysis. However, the vast majority of the nuance in these is irrelevant to finance and

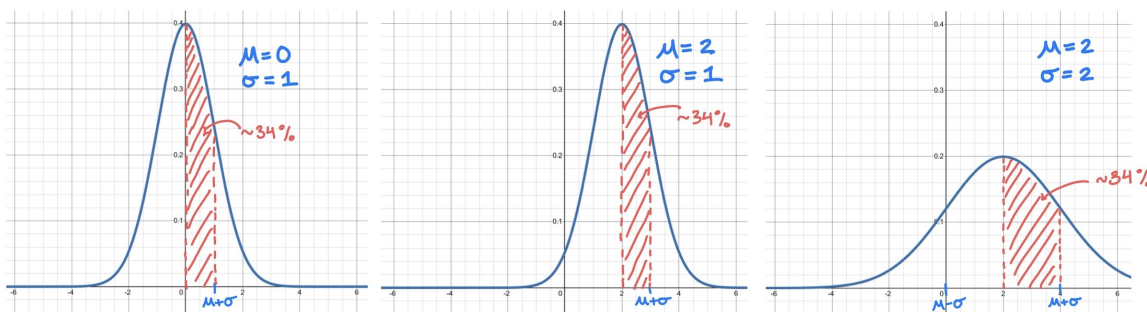
to computer-driven application domains, as minimal units for prices and for computerized computations mean that ultimately everything we are working with is finite. Sometimes, though, it is still convenient to start with a continuous definition, since this doesn't pre-commit to a level of precision/rounding/bucketing.

4.3 Gaussian Distributions and the Central Limit Theorem

With all of this context in place, we can define the most famous probability distribution, the Gaussian (a.k.a. normal) distribution by defining its probability density function for real numbers x as:

$$f(x) := \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

There are two parameters here, μ and σ , that represent the expected value and the standard deviation of this distribution respectively. If we graph this probability density function for various values of μ and σ , we will see the “bell curve” shape we have likely encountered many times before.



As we might guess from these examples, changing the parameter μ moves the center of the bell shape, and changing the parameter σ affects the width of the bell shape. But even as we vary μ and σ , the core shape remains true. For example, the fact that the range from μ to $\mu + \sigma$ represents about 34% of the total probability is true for all values of μ and σ .

You may be used to seeing this shape not as a graph of the probability density function, but rather as the shape of a histogram of the sampled values from a Gaussian distribution. These are essentially the same thing, except that the histogram will be “chunkier” depending on the bucketing, and noisier, depending on the sample size. To see why these are the same basic shape, note that the number of times a value occurs in sampling should be roughly proportional to its probability, so it is a general fact that with enough samples, the histogram of sampled values will converge to the shape of the underlying probability distribution.

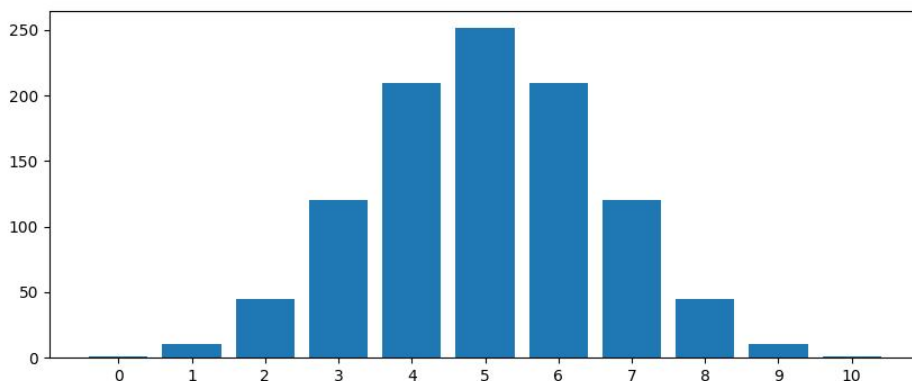
Theoretically, any non-negative function whose integral over a specified range is equal to 1 is a valid probability density function. So why so much fuss over the Gaussian distribution in particular? In statistics class, you may have been told that it’s “naturally occurring:” seemingly capturing a wide variety of real-world phenomena. There is a mathematical reason for this. It’s most succinctly expressed in the *central limit theorem*, a statement about what happens we average a lot of samples that are independently generated from the same underlying probability distribution.

Formally, we can imagine a long line of independent random variables, each with the same distribution: $X_1, \dots, X_n$. A good example to keep in mind is n coin flips. Now we can define the random variable

$$X := \frac{X_1 + X_2 + \dots + X_n}{n}.$$

If we stick with our n coin flip example, let’s assign numerical values to heads (1) and tails (0). Now the numerator in our definition of X represents the number of heads we get in n flips. For a fair coin, it is clear that the expected value of this X is $\frac{1}{2}$: we should get roughly half of our n flips turning out heads, and roughly half turning out tails. But we may not get *exactly* half of each. If we flip a coin 10 times, for example, we may reasonably get 6 heads and 4 tails, or 4 heads and 6 tails, or 7 heads and 3 tails, etc. In fact, every set of outcomes is possible: it is even possible to get 10 heads, or 0 heads. But not every set of outcomes is equally likely: 5 heads and 5 tails is much more likely than 10 heads and 0 tails.

This feels obvious, but it’s worth being explicit about why this is true. Each exact sequence of 10 outcomes here is equally likely (because it is a fair coin), so the sequence (1,1,1,1,1,1,1,1,1,1) (or heads, heads, ... etc.) has the same probability as the sequence (1,1,1,1,1,0,0,0,0,0). However, there is only *one* sequence that has 10 heads, and there are *many* sequences that have 5 heads (other examples include (0,0,0,0,0,1,1,1,1,1), (0,1,0,1,0,1,0,1,0,1), and so on). If we make a histogram of how many ways there are to each get possible number of heads from 0 to 10, guess what it looks like:



Suspiciously familiar, perhaps? It's pretty clunky because we're using a small $n = 10$, but it is starting to flesh out that Gaussian distribution shape. This is what the central limit theorem says - that the distribution of X (an average of n independent samples of the same underlying distribution), converges to a Gaussian distribution as n gets larger. The expectation of that Gaussian distribution is the same as the expectation of the underlying distribution, and the variance of that Gaussian distribution is the variance of the underlying distribution divided by n . [This inverse dependence on n makes sense since the distribution gets more and more concentrated around its mean as the sample size grows.]

The shocking part is - it doesn't matter what the underlying distribution is. It can be anything! But the key caveat here is that the Gaussian magic only applies to the *average* of the many samples, not the samples themselves, and not to some arbitrary function of them.

Proving the central limit theorem is a bit of a pain. It's worth reading through once, ideally with a glass of wine in hand and a ready quip like "but I can't do the dishes now dear, I'm proving a *theorem*!" But the main thing to know is that it's very general (did we mention it holds for *any* underlying distribution?!) while also having some important caveats (it's specifically for averages and assumes identically distributed independent samples, though there are some more specialized versions that weaken this requirement in various ways).

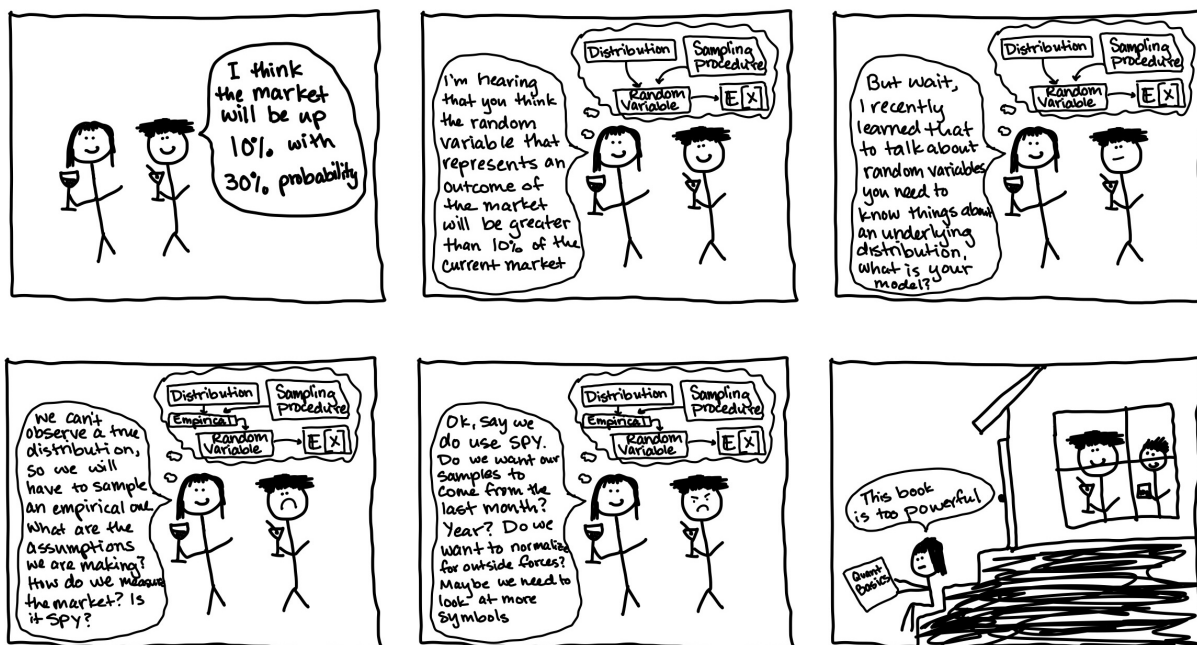
4.4 Probability Distributions in the Natural World (and Financial Markets)

So what does all of this have to do with financial data? You rightfully ask. After all, there's no explicit random process involved in stock trading. We don't flip coins or roll dice to decide which stocks to buy or which prices to set. (We could, though we typically wouldn't admit it.) Instead, analyzing financial data through the lens of probability theory belongs to the rich (and sometimes fraught) tradition of treating complex, external systems as "random" rather than as simply "unknown" or "incompletely understood." Physicists have argued, in fact, about whether *all* so-called random processes are really just models of our incomplete knowledge of deterministic systems. Such arguments, heated in the early era of quantum mechanics, inspired the famous Einstein quote that "God does not play dice." [Spoiler alert: apparently She does?] But such arguments are well beyond our expertise, except to say that the notion of true randomness in the world appears to have been vindicated.

If we say something like "the probability of this stock price going up by ≥ 1 dollar in the next hour is...", we are treating the stock price as a random variable, even if we are not being sufficiently explicit about what its probability distribution or its sampling mechanism is. It

is useful to say things in shorthand like this, as carrying around the probability distribution explicitly all the time can be clunky and inefficient. However - if we, when pressed, cannot define the probability distribution we have in mind - then we are not making a statement of probability theory. We are rather appropriating the language of probability theory to make an unscientific statement sound more authoritative, and this is an insidious habit. Here is a game you can play that will make you profoundly unpopular at dinner parties: whenever anyone says “the probability of something is ...”, you furrow your brows in a look of thoughtful interest and sincerely ask, “under what probability distribution?”

This is something always worth thinking about though - what actual process is probability being used to model here? How is the model formed? And does the model make sense? A basic default is: thinking about a new uncertain event as having the same distribution as a recently observed empirical distribution. If we casually say “the chance of her being on time is about 30%,” we can unusually unpack this as something like: “In similar situations I’ve observed recently, she was on time about 3 times out 10.” We are side-stepping the issue here of what drives her to be on time for some of these times and not others, preferring to mentally model her behavior as “random.” Note though, that the line between what we model as random and what we attempt to account for deterministically is fuzzy. The “in similar situations” part still leaves us with a range of options for how we organize our observations. There is an inherent tension here: we may hope that by being more restrictive in what counts as a “similar situation,” we are focusing on the *most relevant* data points, and thus improving the quality of our probability estimate. But we are also making the data set smaller, thus relying on fewer and fewer examples from which to estimate an average.



At this point, you may be wondering why we've been talking for several pages about something that most quantitative research papers would absorb into an opening phrase: "Consider the market as a random variable X whose distribution ... " But it's important to know just how many slights of hand go into constructing such a phrase. It's the kind of phrase that makes us feel like we are being more scientific, more objective. And in some ways we are! But it can also obscure a lot of subtle and somewhat arbitrary decisions that are now baked into our reasoning.

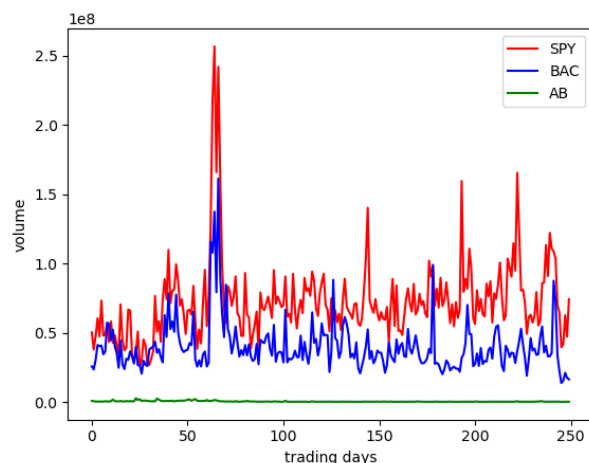
It isn't really an option to avoid probability distributions in reasoning about financial markets. Whether it's intrinsic to the world in the way of quantum mechanics, or merely a consequence of practical constraints, there are always going to be fluctuations in financial data that defy our most comprehensive attempts to deterministically explain them. But the shifting lines between what we *try* to describe deterministically and what we relegate to the realm of randomness is a degree of freedom that we must constantly re-examine and renegotiate. If you ever find yourself needing (or wanting!) a deeper background in the kind of computationally effective probability theory that is most often relevant to computer science and finance applications, we highly recommend [14] as a general resource.

Chapter 5

Data Normalization, or how to read one story with thousands of characters

Now that we’ve thought abstractly about probability distributions, let’s think about the process of taking financial data and viewing it as samples from probability distributions. There are challenges to this. For one, trying to make sense of stock market data overall is a bit like trying to watch a play where every character is speaking in a different language. Each symbol has its own typical price range, volume, and volatility, and this is important context for interpreting its behavior on any given day. We could treat the overall data as the union of 1200+ disconnected stories about individual symbols, but this would blind us to general trends and prevent us from accumulating sample sizes across symbols. When possible, it is better to “normalize” data over symbols so that it can be blended together in a more apples-to-apples way.

For instance, let’s suppose we are trying to study patterns in volume. Here is a plot of the volume in each trading day in 2025 for three symbols: SPY, BAC, and AB:

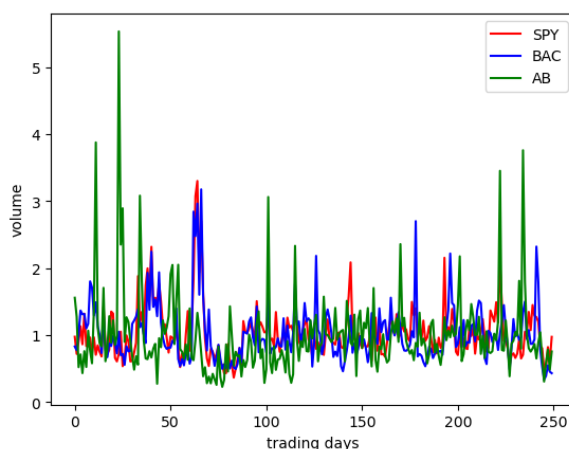


The x-axis here is just counting up in trading days, and the y-axis is in units of shares traded. The red line here is SPY, the blue line is BAC, and the green line is AB. These symbols are somewhat arbitrary choices: we chose SPY and BAC because they are both very liquid, so it happens to be that you can visually look at them at the same scale and make some sense of the combination. We chose AB for contrast, as it is not nearly as liquid, and hence you can't really see its' meaningful movements when you try to plot it on this same scale. (Also, AB are Allison's initials.)

There are various methods available to gain information across symbols with different characteristics. Selecting an appropriate method depends on what the differing characteristics are and what outcomes you are trying to understand. We outline a set of useful normalizations in the following sub-sections.

5.1 Adjusting by Average Daily Volume (ADV)

To put raw volume numbers in context and make comparisons between symbols more meaningful, we can compare them to recent averages or other reference values for each symbol. A commonly used reference is the average daily volume (ADV), which is computed by averaging the daily volume over a specified time period such as 20 or 30 days (could be trading days or calendar days, the results are similar). If we look at each day's volume as a percentage of a rolling ADV, we get a quantity that is more meaningful to work with across symbols. To make the chart below, we took the volume for each symbol on each day and divided it by the ADV for that symbol, which we computed by averaging over the previous 20 trading days:



The colors in this chart are the same as before: the red line is SPY, the blue line is BAC, and the green line is AB. The y-axis is now in multiples of the ADV for each symbol, as

of each day. You'll notice something interesting happened - the green line that was flattest in absolute terms became the most volatile in relative terms. This is not too surprising, as smaller numbers/smaller sample sizes often exhibit more volatility than larger ones. Now that everything is on a more comparable scale, we might also see some correlation between the shapes of these different curves, though of course there are individual phenomena and noise at play as well.

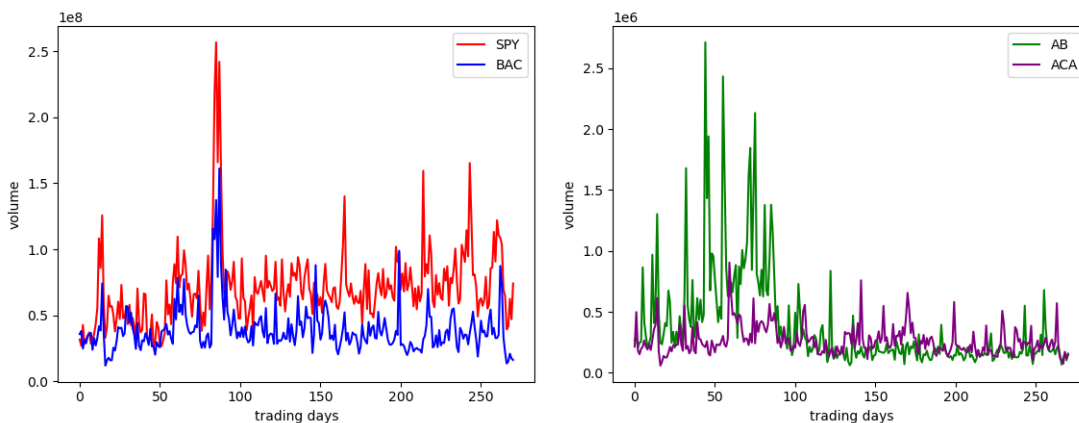
5.2 Adjusting to Relative Price Movements

If we are working with prices, we can similarly divide by a reference price to convert to ratios that are more comparable across symbols. We might divide the last price over a time interval by the first price in a time interval, for example, to convert absolute movements in price into relative ones. This process of normalization can have multiple layers, depending on what we are interested in. We could further put relative price movements in context by dividing them by some recent average of volatility for that symbol, if we are interested in gauging how typical or atypical a percentage price movement may be. This is a bit like taking derivatives in calculus or physics - we can look at an absolute change in price, or a rate of change, or a rate of a rate of change, etc.

5.3 Grouping

If we don't want to divide directly by a reference value in order to normalize our data, we might consider organizing our data into groups instead. Creating groups of symbols that have similar ADVs, for example, might have similar effects to dividing by the ADV. One nice thing about grouping is that it doesn't commit to an exact mathematical relationship between the raw values we care about and the quantity we would otherwise divide by. One not nice thing about groups is that partitioning our data makes our sample sizes effectively smaller. For this reason, we should use groupings sparingly.

Here is a side by side of volume charted separately for two groups of symbols, one that contains the very liquid symbols BAC and SPY, and one that contains the less liquid symbols AB and ACA (chosen because it has a similar volume to AB through at least parts of 2025):



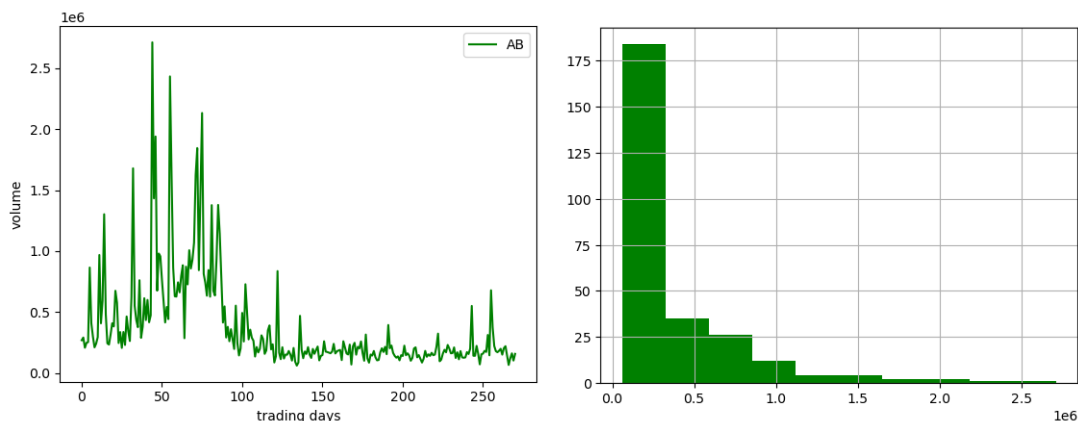
Putting these into groups makes it easier to choose a scale for the y-axis that makes our visual representations more meaningful and interpretable by eye. We might also use such groups in further analyses if we want to train different models for different symbol groups. This is typically most useful if we suspect the nature of a phenomena we are trying to capture is qualitatively different across groups, rather than primarily differing in scale.

5.4 Re-Centering

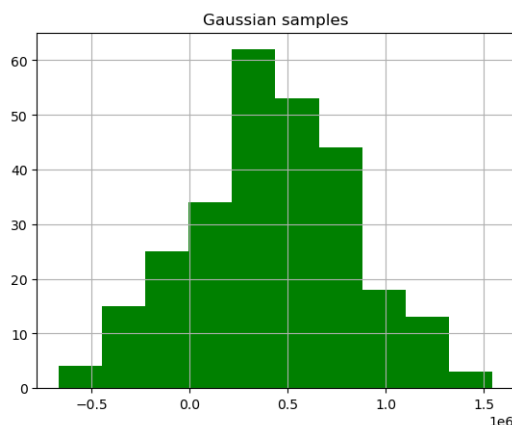
Sometimes we may want to “re-center” our data, especially if we are viewing it as a probability distribution. This could mean subtracting the average or median from a set of data points to make the resulting data points “centered” around 0. We can also divide everything by the standard deviation, which forces the new standard deviation to be 1. Together, these adjustments can guarantee that we are working with a distribution of data points that has mean 0 and standard deviation 1, like the normal Gaussian distribution $N(0, 1)$. This kind of normalization can be very useful if we want to understand the relationships between distributions, and we want to look beyond surface differences in means and variances. There are two important things to keep in mind when doing this: 1. just having mean 0 and standard deviation 1 *does not* imply that a distribution is $N(0, 1)$. If you start with data that does not fit a Gaussian distribution, it won’t fit after these adjustments either. And 2. these adjustments can only be computed *after* all of the data is collected, not known in real-time as the data is being collected (unlike adjustments like dividing by the ADV as of yesterday). Such adjustments can muddy the waters considerably when we want to think about causality, and look-ahead bias can contaminate our results.

Let’s see what this looks like for daily volume numbers for the symbol AB over the year 2025. This data set contains 271 numbers, each representing the number of shares of AB that traded on a particular trading day in 2025.

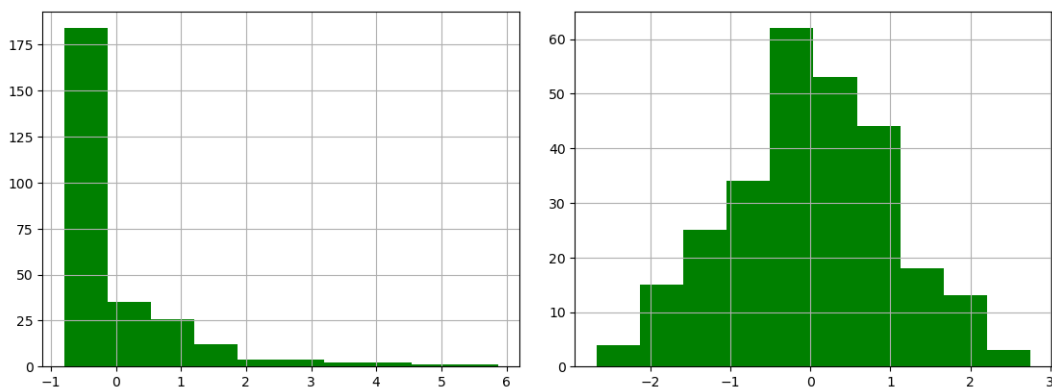
Here's what the raw numbers look like, both as a time series and as a histogram:



The empirical mean here is roughly 376,282, the median is 211,390, and the standard deviation is 397,956. There are several ways in which this distribution substantially deviates from the Gaussian template. The mean and median are substantially separated, the distribution does not appear symmetric around the mean, and the high standard deviation may be strongly driven by outliers rather than a more general affect. If we sample 271 values from a Gaussian distribution with the same mean and standard deviation, we can see the contrast. Here's what a Gaussian distribution with these parameters and this sample size looks like:



Let's now adjust both distributions (the real one and the sample Gaussian one) by subtracting the mean and dividing by the standard deviation. We'll look at the results side by side:



The adjustments here made the scales for these graphs more compatible, but it did nothing else to reconcile their disparate shapes. This is an important picture to keep in mind when we talk about “normalized” data - the name is fundamentally misleading. The “normal” distribution that it evokes (a Gaussian distribution with mean 0 and standard deviation 1) is much cleaner and prettier than our real distributions are likely to be. In this particular case, this shouldn’t be surprising, and it’s important to note that viewing the daily volume numbers over a year as independent samples from a probability distribution is not particularly meaningful. We don’t really imagine that the market is sampling a volume number from such a distribution each day, oblivious to what has happened recently, what day of the year of it is, and what is happening in other symbols, etc. The only sampling mechanism that would realistically produce this distribution would be: “pick a random trading day and look up the volume on that day.” This example is just meant to be a simple illustration of data normalization - modeling various market dynamics more realistically as probability distributions requires more nuance, and we will get into that more later.

Overall, data normalization can be a powerful tool to allow us to pool data meaningfully across symbols or other parameters, as well as to help us reduce the effects of market noise. Data normalization in some form is often a necessary pre-processing step in building models with financial data. But it should be done with care, as it is not a panacea, and bugs in data normalization and/or poorly chosen adjustments can render data useless or misleading.

Chapter 6

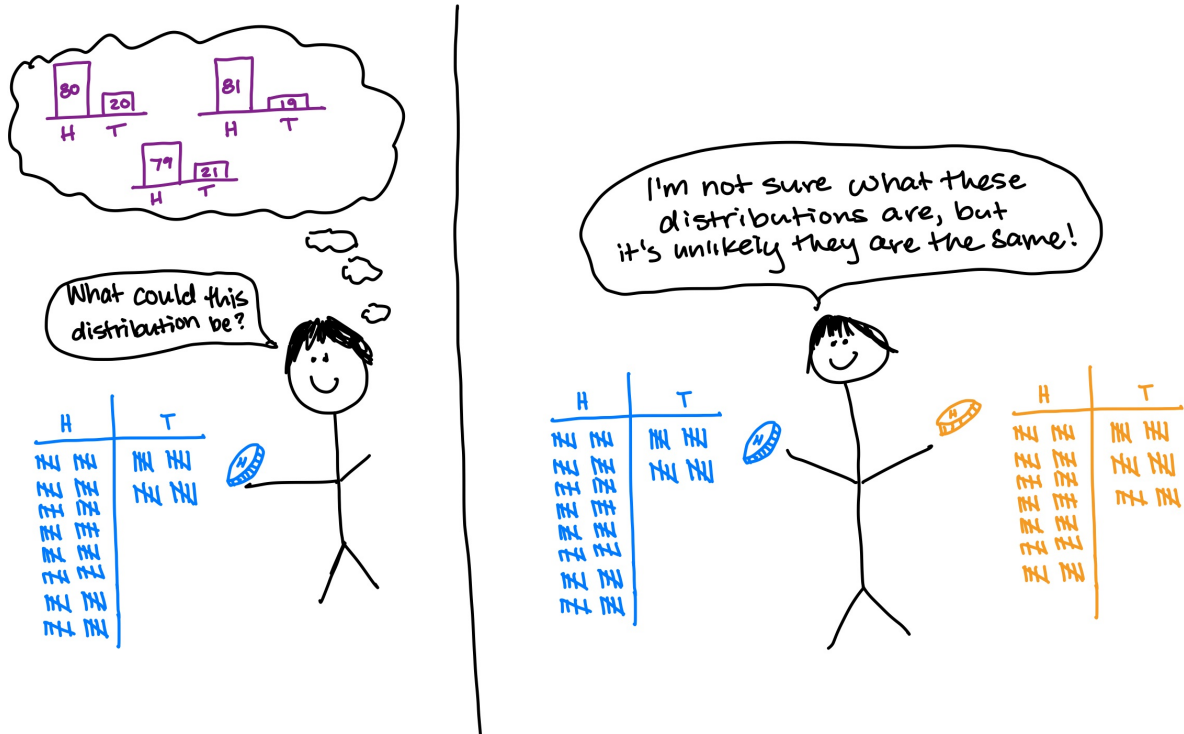
Distribution Testing

Since we need probability distributions to reason about financial data, it follows naturally that we will need good ways of telling them apart. Is my data roughly consistent with a Gaussian distribution? we may ask (as we did above). Or - do these two sets of observations appear to be coming from the same distribution, or from two wildly different ones? Such questions are in the realm of *distribution testing*, a subfield of property testing in computer science/statistics concerned with learning what we can about unknown probability distributions from their samples.

It should be intuitive that we cannot hope to tell apart distributions that are extremely similar, unless we have an absurdly high number of samples. Suppose, for example, that a distribution D_1 places probability 0.00001 (i.e. a 1 in 100,000 chance) on a particular event, while a different distribution D_2 places probability 0.00002 (a 2 in 100,000 chance) on the same event. Now, if we get 100 samples of each distribution, then with very high probability, this event won't occur at all in either case. So we won't be able to "see" the difference between a probability of 0.00001 and 0.00002. This phenomenon generalizes to the fact that the precision of our probability estimates is limited by our sample size, even for very common events.

When we try to recognize or learn about a distribution from a fixed set of samples, the information we're getting is inherently partial and imprecise. To interpret this correctly, it helps to keep a mental map of the set of probability distributions we are considering, and to define a distance metric for distributions. Distributions that are "close" to each other under this metric are going to be hard to tell apart, and those that are "far" are going to be easier. For example, suppose that we have 100 samples of a coin being flipped, and we know the coin is biased, but we don't know by how much. Let's say in the 100 samples, there are 80 heads and 20 tails. Now suppose we have 100 samples of a different coin being flipped, and this is one is 30 heads and 70 tails. We may not be very confident that the bias of the first

coin is 80/20 as opposed to 81/19, for example. But we can be quite confident that the two coins have very *different* biases. [If they had similar biases, these widely disparate outcomes would be highly unlikely.]



A natural distance metric for probability distributions is total variation distance, which can be defined for discrete distributions by:

$$d(D_1, D_2) = \frac{1}{2} \sum_{x \in \Omega} |D_1(x) - D_2(x)|.$$

Here, Ω denotes a finite set of possible outcomes, while $D_i(x)$ denotes the probability of outcome x under distribution D_i . An equivalent way to define this is:

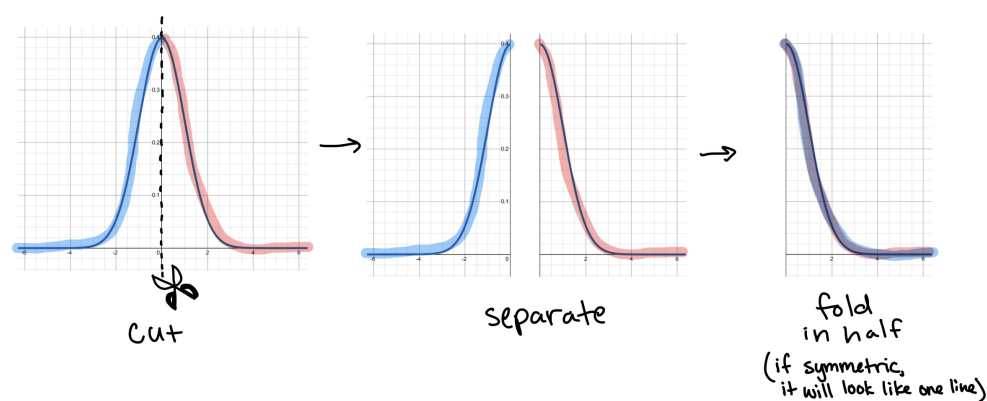
$$d(D_1, D_2) = \max_{S \subseteq \Omega} (D_1(S) - D_2(S)),$$

where $D_i(S)$ denotes the probability that D_i places on the set of outcomes in S , and we are maximizing over all subsets of possible outcomes.

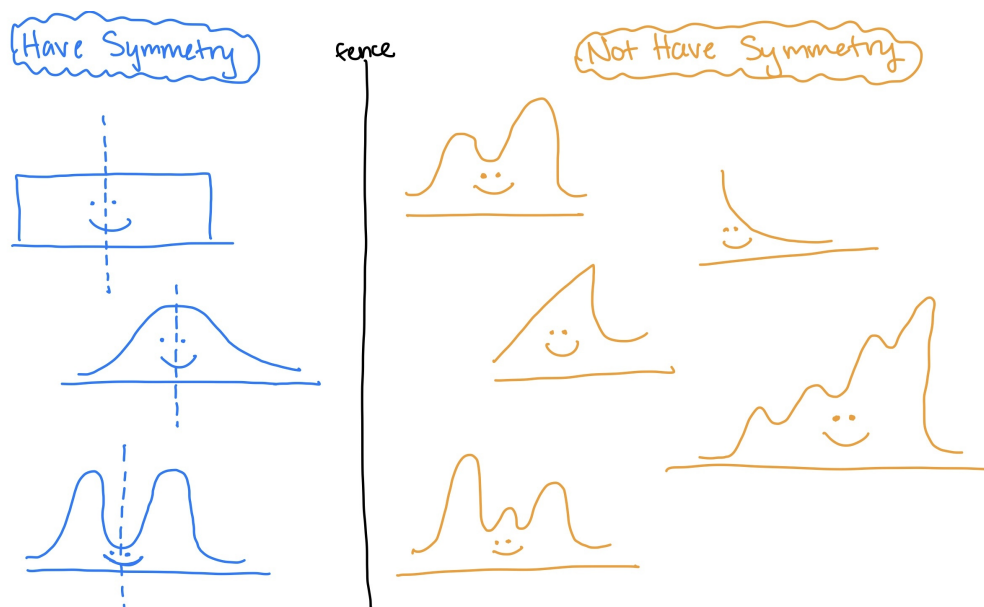
Let's see what this metric has to say about our coin-flipping example. A coin with bias 80% to heads and a coin with bias 81% to heads would have total variation distance of 0.01, as this represents the difference in the probabilities placed on heads and tails. A coin with bias 80% to heads and a coin with bias 30% to heads would have a total variation

distance of 0.5. In this sense, the metric matches up with our intuition: it places the coin-flipping distributions that have similar probabilities on heads much “closer” to each other than distributions with markedly different probabilities. We can roughly think of getting more samples from a distribution as a process of gradually narrowing down its location on the implied geographic map of distributions that we get when we work with a metric like total variation.

We may casually think of a “property” of a distribution as a quality that a distribution can have or not have. Like symmetry: a Gaussian distribution is symmetric around its mean. In other words, if we graph the probability density function of a Gaussian distribution and slice it down the middle, we could fold one half over the other and it would perfectly match:

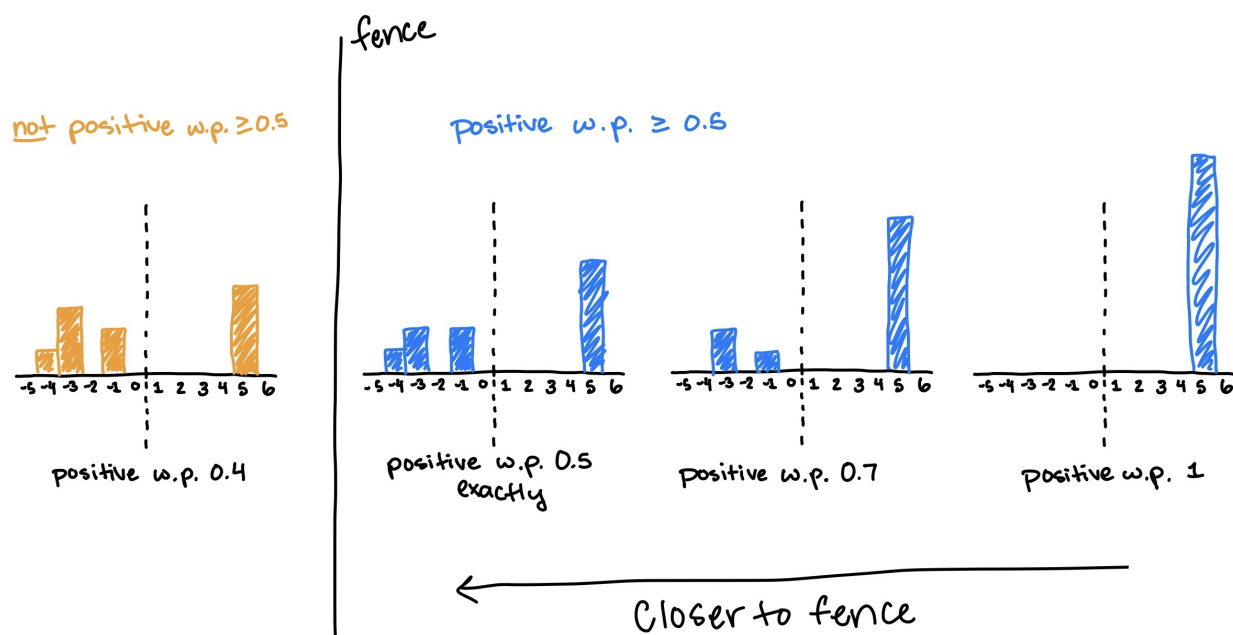


An equivalent way to think of a property is that it divides the space of probability distributions into “haves” and “have-nots.” We could imagine putting up fences in probability distribution land to separate the distributions that have the property (e.g. symmetry) from those that do not.



If we then imagine getting rough geographic information about a distribution by looking at some samples, we might hope to be able to say which side of the fences our distribution falls on. How hard or easy this is depends on how pathological the fences are: fences that weave in and out in close proximity will make it hard for us to be confident which side our distribution falls on.

For example, let's consider a property like “takes a value > 0 with probability at least 0.5,” in the land of probability distributions for random variables that take values in the range $[-10, 10]$. Some things are squarely on one side of the fence - for example, a distribution that puts all of its weight on the value 5. Anything close to this distribution in our metric of total variation will also be squarely on the “has property” side of the fence. If we start moving incremental amounts of probability mass away from the value of 5 and onto negative values, it will take us quite awhile to get to a distribution that does not have the property in question. We can roughly think of this as having to walk a long distance to reach the fence itself.



This sort of thing bodes well for being able to test whether an unknown distribution has the property. Nonetheless, it's always possible that the unknown distribution lands close to the fence. For example, suppose we have a distribution that takes the value 1 with probability 0.5 and the value -1 with probability 0.5. It would only take a razor thin deviation from this to reach a distribution that does not have the property.

For some properties, nearly every distribution might be “on the fence,” as it were. Consider, for example, a property like “the hundredths digit of the probability of the outcome

0 is odd.” Then *any* distribution that has this property is very close to a distribution that doesn’t - just by changing the probability mass on the outcome 0 by ± 0.01 .

Let’s return to our example of symmetry around the mean as a property. For a distribution that has this property, a tiny change in probability mass would move it over the fence. Symmetry is a fragile thing - a small deviation from it can make things asymmetric. Thus it won’t be possible to look at a reasonable number of samples of a distribution and say with any confidence: “this distribution is symmetric.” But it would be possible to say with reasonable confidence: “this distribution is rather close to being symmetric” or “this distribution is rather far from being symmetric.”

Generally, the goal of distribution testing is to be able to tell the difference between distributions that have a property and distributions that are *far* from having the property (in the sense that any distribution having the property is far away in total variation distance). For a property like symmetry, this goal makes sense, and is achievable. For a property like our pathological example, “the hundredths digit of the probability of the outcome 0 is odd,” there aren’t any distributions that are *far* from having the property, so there isn’t really anything meaningful that we can do. [Aside: you might wonder - what if we want to tell the difference between distributions that are *close* to having a property and distributions that *far* from having that property? This is called “tolerant” testing, and is often possible too.]

The field of distribution testing is substantial and active, so it would be an untenable digression to cover a significant portion of it here. If you are interested, a more devoted resource for this topic is the survey [9]. For now, we’ll focus on the basics so you can get of sense of how the mindset of distribution testing can be useful in finance.

6.1 Uniformity Testing

Let’s start with a basic question in the realm of distribution testing: is this distribution uniform? In this case, we are trying to distinguish between a uniform distribution and a distribution that is *far* from uniform. How might we do this, and how many samples should we expect to need? This was first considered (implicitly) in [10].

A key idea here is that we can look at how many collisions we have among our samples. In this context, a collision refers to a pair of samples that have the same value (e.g. two coin flips that are both heads, or two dice rolls that are both 6s). To see why the collision count is helpful, let’s start by considering the probability that a particular pair of samples take the same value. If Ω represents a finite set of possible outcomes, then that probability that a random variable X_i and a random variable X_j take the same value when they are both independently distributed according to a distribution D is:

$$\sum_{x \in \Omega} \mathbb{P}[X_i = x] * \mathbb{P}[X_j = x].$$

A term here like $\mathbb{P}[X_i = x] * \mathbb{P}[X_j = x]$ is just the square of the probability that distribution D places on the outcome x . So the expression above is just the sum of the squares of all the individual probabilities that make up the distribution D . In typical math notation, we can write this as:

$$\mathbb{P}[X_i = X_j] = \|D\|_2^2,$$

since $\|\cdot\|_2$ is a notation for the ℓ^2 -norm, which just means taking the sum of the squares of the individual components of a vector.

Let's think about what kind of distributions have relatively high and low values of $\|D\|_2^2$. Fundamentally,

$$\|D\|_2^2 = \sum_{x \in \Omega} p(x)^2,$$

where $p(x)$ denotes the probability that a distribution D places on the outcome x . So $\|D\|_2^2$ is a sum of squares of probability values, which themselves are non-negative values that sum to 1. In an extreme case, suppose $p(x^*) = 1$ for some particular value x^* , and $p(x) = 0$ for all $x \neq x^*$. Then $\|D\|_2^2$ would be 1. This is actually the highest value of $\|D\|_2^2$ that we can get! This makes intuitive sense when we view $\|D\|_2^2$ as the probability that two independent samples of D turn out to be equal. If D is just a constant and always spits out x^* , then the samples will be the same with probability 1.

Now let's consider what happens as the probability starts getting more spread out. Suppose we have non-zero probability values of $\frac{1}{2}$ each on two particular outcomes. Then:

$$\|D\|_2^2 = \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2 = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$$

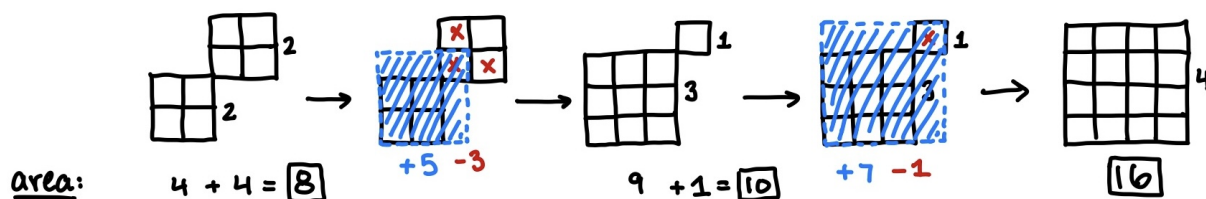
This is actually the smallest value we can get for any D that only takes on two values. Compare this, for instance, to a D that places probability $\frac{1}{3}$ on one value and probability $\frac{2}{3}$ on another:

$$\|D\|_2^2 = \left(\frac{1}{3}\right)^2 + \left(\frac{2}{3}\right)^2 = \frac{1}{9} + \frac{4}{9} = \frac{5}{9}.$$

Again, this is fairly intuitive: two flips of a biased coin are more likely to be equal to each other than two flips of a fair coin.

What's happening here is kind of fun to visualize geometrically. For convenience, let's use whole units instead of fractional ones for the visualization. Suppose we want to make two squares such that the sum of their side-lengths is equal to 4. There are three different

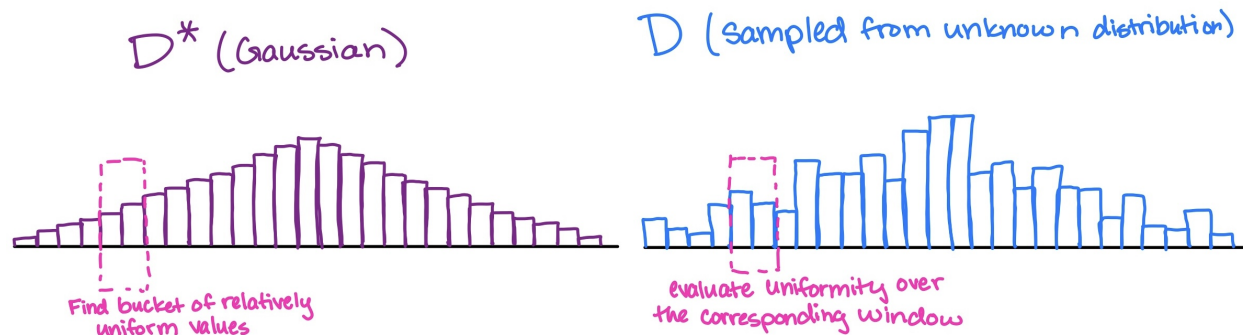
ways that we could do this: two squares of length 2, one square of length 1 and one of length 3, or a single square of length 4.



Notice what happens as we go from two squares of length 2 to a square of length 1 and a square of length 3: the total area covered by the two squares grows. We can see this because adding another “layer” to make a square of 2 into a square of 3 adds 5 new units of area, whereas downgrading a square of 2 to a square of 1 subtracts only 3 units of area. Thus, whenever we have non-negative numbers whose sum has to be fixed, the *least* we can get for the sum of the squares will come from having each individual value be as similar as possible. If there are two values where one is much bigger than the other, we can lower the total sum of squares by re-balancing them.

Putting all of this together, we have found that $\|D\|_2^2$ is smallest whenever the distribution D is closest to uniform, and largest whenever it is fully concentrated on one value. Also, $\|D\|_2^2$ is somewhat easy to estimate, since if we take m independent samples from the distribution D , the number of collisions we expect is $\binom{m}{2}\|D\|_2^2$. Because $\binom{m}{2}$ is quadratic in m here, it’s not too surprising that we can get a decent estimate of this with a number of samples that is proportional to $\sqrt{n}$, where n is the size of the domain (i.e. the number of possible outcomes). Furthermore, it turns out that the total variation distance between D and the uniform distribution can be bounded in terms of $\|D\|_2^2 - \frac{1}{n}$, where $\frac{1}{n}$ is the value of $\|U\|_2^2$ where U denotes the uniform distribution on n elements.

Let’s take another example: what if you want to know whether your unknown distribution D is the same as some known distribution D^* ? Maybe D^* is Gaussian, for example, or an empirical distribution built from data you’ve collected on a different time period. Even when D^* is not uniform, it turns out we can approach this problem using the same tools we used to test uniformity, if we just break up the set of outcomes into pieces where D^* is approximately uniform. This core idea is originally developed in [4]. More specifically, we’ll define “buckets” of outcomes, where the probabilities assigned to outcomes within the same bucket are all within a tight constant factor of each other.



The nice thing about this is that we only need a logarithmic number of buckets in terms of the precision of probabilities we want to achieve, so we've now reduced our problem into a set of subproblems we already know how to solve (testing approximate uniformity within a bucket), as well as a problem over a much smaller domain (testing that the distribution over buckets agrees with D^*). Once the domain is much smaller, testing things about distributions becomes much easier, since you will now have many more samples per outcome (for reasonably common outcomes at least). For distributions with a small number of outcomes and a lot of samples, we can basically plot their histograms and compare them point-by-point. Any glaring differences on common outcomes will come to light this way. We could quantify this by computing the total variation distance, or any other metric we choose. It's important to note that tools like comparing histograms and looking for collisions only make sense after pre-processing/rounding/bucketing more continuous distributions, as more continuous distributions won't naturally repeat the same exact values very often.

We're not going to go deeper into distribution testing here, but we hope that this gives you a sense of what it entails. This should allow you to recognize when something you're working on intersects with the field of distribution testing, and in such cases you can go dig deeper into particular distribution testing tools that may be relevant for your problem. For example, there are specialized tests for cases like testing if your distribution is Gaussian (e.g. the Shapiro-Wilk test). There are many ways in which distribution testing might come up in financial analyses. For example, suppose you are interested in looking at the distribution of timestamps of trades. Is the second value of these timestamps uniformly distributed, you might ask, or is there some bias towards trading near the beginning or end of a minute boundary? Collision counts could help you here distinguish a uniform distribution from a non-uniform one. Similarly, distribution testing could help you look for distinguishing characteristics of your own trading activity versus the general noise of the market.

Chapter 7

Hypothesis Testing, Monte Carlo Simulations, and when it's ok to skip the math

There's a form of question that drives a lot of people into looking for statistical evidence, and it's deceptively basic: "I believe [fill in the blank]. Does the evidence support this?" It's not hard to see why people want to frame statistics this way, and at a first glance, descriptions of classic hypothesis testing seem to align with this framing. We have a hypothesis that X causes Y, for example, so we ask - is there statistically significant evidence of this? The problem is, this question is nonsense. It sounds like it means something, but it really doesn't. What counts as evidence? What counts as statistically significant? Both of these notions rely on much more context than our seemingly straightforward question is equipped to provide.

Instead, we like to think of hypothesis testing as driven by a different question, one that gets taught frequently in training for improv or sketch comedy: "If this is true, what else is true?" In comedy, this question guides the exposition. It isn't usually very funny to take a bunch of disjointed weird things and stick them together. Better comedy often comes from taking one weird thing, and then following its natural consequences. Like - if aliens abduct a soccer mom, she's probably going to ask if everyone remembered to go to the bathroom before the spaceship takes off.

In retrospect, this is the mindset that drives distribution testing as we discussed in a previous section. "If my data is drawn from a uniform/Gaussian/etc. distribution then ..." - whatever follows is what we then test for. Hypothesis testing more broadly is similar. We think of a consequence of our hypothesis - one that happens if our hypothesis is true, but does not happen in a specific alternative scenario where our hypothesis is false. We then look for that consequence to see if we get strong evidence one way or the other.

Let's suppose we want to test the hypothesis: buy orders have less average slippage than sell orders. We can obviously compute the two averages, one over a set of buy orders and one over a set of sell orders, and suppose we see that the average for buy orders is less by 2 bps. [Note: recall from Chapter 3 that bps ("basis points") are 1/100th of a percent.] But this alone isn't very meaningful - afterall, the buy and sell orders are best thought of as sampled from distributions, and observing that an empirical average of samples is higher than another empirical average of samples does not necessarily tell us much of anything about the underlying distributions.

We might model our two slippage distributions as Gaussian, and ask: if these distributions in fact have the *same* mean, what's the probability that their samples produce empirical means that are at least 2 bps apart? The hypothetical world here where there is no meaningful difference in means is called the *null hypothesis*. In the case of Gaussian distributions, we can answer this kind of question precisely if we have good estimates for the relevant variances. In the simplest case, let's suppose the variances are equal, and it is only the means that may differ.

Let's write this out in formal math notation. Let $X_1, \dots, X_n$ represent the slippage values (in bps) of our buy orders, treated as independent Gaussian variables, and let $Y_1, \dots, Y_m$ denote the slippage values of our sell orders, similarly treated as independent Gaussian variables. We are presuming here that all X_i 's have the same expectation as each other, and all Y_i 's have the same expectation as each other, so let's call these expected values μ_X and μ_Y respectively. In our hypothetical world (a.k.a our null hypothesis), $\mu_X = \mu_Y$.

Our empirical estimate of the difference in means (assuming for simplicity that we weight all orders evenly - more on this later) can be written as:

$$\frac{X_1 + \dots + X_n}{n} - \frac{Y_1 + \dots + Y_m}{m} = \frac{1}{n}X_1 + \dots + \frac{1}{n}X_n - \frac{1}{m}Y_1 - \dots - \frac{1}{m}Y_m.$$

We can see here that this is just a linear combination of independent Gaussians, and thus the distribution of this quantity is also Gaussian. Its expectation is $\mu_X - \mu_Y$, and its variance is twice the variance of the original X_i 's and Y_i 's. Under our null hypothesis, $\mu_X - \mu_Y = 0$, and we can calculate the probability that a 0-centered Gaussian with this variance will spit out a value that is $\geq 2bps$. If this value is reasonably high (e.g. say 30%), we don't have sufficient evidence to reject the null hypothesis. In other words, what we're seeing is reasonably consistent with a world where there is no difference between the means of X_i and Y_i . If the value of the probability instead is implausibly low (e.g. say 0.01%), then we have very strong evidence that we *do not* live in a world where the means of X_i and Y_i are the same, and we reject the null hypothesis.

7.1 Pitfalls and Replication

This probability of seeing an equal or more extreme result in the hypothetical world of the null hypothesis is called a *p-value*, and the standard notion of “statistical significance” requires a p-value of $< 5\%$ (though sometimes 1% is specified instead). A popular interpretation of statistical significance is that it means “this result is unlikely to be due to chance.” However, there are a lot of structural pitfalls here to be aware of. Some relate to faulty practices we may fall victim to in our research, while others relate to the external structure of scientific evaluation or publication.

The first pitfall is not accounting for the number and frequency of experiments we perform. Say, for example, we want to test a null hypothesis for several stocks individually - perhaps hundreds or thousands. In a world where the null hypothesis is true for every stock we test, a p-value threshold of $< 5\%$ will cause us to falsely reject the null hypothesis for about 5% of the stocks we test. This phenomenon runs counter to the human instinct for interpretation. If we say, “I tested thousands of stocks and this is the most significant effect I found,” we may be tempted to give the result *more* credence rather than less. Wow, the most significant out of thousands? Must be meaningful, right? Actually, it’s probably the opposite: searching in enough places nearly guarantees that we will find some coincidences that register as significant. The same applies when we repeat the same tests over and over again on different time periods - eventually an “unlikely” event is bound to happen merely by chance.

A second pitfall occurs in our interpretation of results that are brought to us, rather than arising from our own research. In many cases, it is impossible for us to account for the *process* that produced such results, as it may be opaque to us. We may not be told how many similar experiments were performed before the supposedly “significant” result was found. We may not be told that attempts at replicating the result failed. A version of this pitfall appears in the scientific publication process, as studies that produce surprising results might be published, while those that produce more boring results may be rejected.

Taken altogether, this means that there are abundant opportunities for null hypotheses to be tested again and again in slightly different guises. From these many tests, coincidentally “significant” results are bound to arise. And then human psychology combined with opaque, structurally biased, or nefarious processes for disseminating results, may tend to amplify surprising or appealing falsehoods. An excellent development of this reasoning can be found in “Why Most Published Research Findings Are False” by John P.A. Ioannidis [11], an article that helped launch what has been termed “the replication crisis” in research disciplines that have relied upon statistical significance as a standard for evaluating proposed findings.

The very name of “the replication crisis” suggests one path towards its resolution: we should insist on replicating any potentially meaningful findings multiple times before we put any trust in them. We can think of this sifting out the coincidences, which should have a low probability of recurring when we test a hypothesis again on new, unrelated data (e.g. from a new time period that doesn’t overlap with our original test). Ideally, true findings should have a high probability of recurring when we test them repeatedly, so replication is a powerful tool for separating spurious results from meaningful ones.

It is worth comparing replication as a strategy to the alternative of simply tightening our standard for “statistical significance.” Let’s say, for example, that we want to test a hypothesis individually in each of 100 stocks, and we declare a result as “significant” if it has a $\leq 5\%$ chance of occurring under the null hypothesis. Let’s suppose, for example, that for 2 of our stocks, the null hypothesis really is false, and the true value for whatever we are measuring is something that would occur under the null hypothesis with probability 4%. Let’s suppose for all of the other 98 stocks we are testing, the null hypothesis is true. If we run our tests and flag everything that falls under our 5% significance threshold, we should then expect to see around $2 + 0.05 * 98 \approx 7$ flagged results. Let’s suppose we get exactly 7 results, containing 2 true positives and 5 false positives. Let’s say we run the tests again on a new data set for these 7 stocks, where the null hypothesis continues to be true for the false positive stocks and continues to be false for the true positive stocks. Then both of the true positive examples have a high chance of surviving this replication test and being flagged again, while the 5 false positive stocks have only a low chance. Collectively, we expect only $0.05 * 5 \approx 0.25$ (e.g. less than 1) false positive after replication. However, if we had set our original threshold of 5% on our original data set to be lower than 4% in order to reduce the false positive rate that way, we may missed the true positive examples. However, we could view the joint process of testing and replicating as simply one testing process with a stricter evidence standard. So one way to think about this is that replication suggests a specific form of hypothesis testing that allows us to set stricter standards without missing too many true positives. The downside here is that replication can require a lot of data, and/or a lot of stability in the phenomena being tested so that it holds up across the replication phase.

7.2 Monte Carlo Simulations

Sometimes it’s not easy to compute exactly what the probability of an event is under a null hypothesis. Luckily, there is a general and flexible tool available to estimate such probabilities, namely *Monte Carlo Simulations*. This tool broadly applies when we have a convenient sampling method for the probability distribution we are trying to understand.

We'll explain Monte Carlo Simulations first by using an example, and then we'll take a step back and think more abstractly. Let's imagine we're looking at a bunch of timestamps for a data set of trades over a symbol within a single day, and we're wondering if there's a weird pattern to them. Suppose we have 1000 trades, and 30 of them occur with a seconds value equal to 0 (meaning that they occur within the first second of a minute). Is this weird? Is it consistent with a null hypothesis that the second value is uniformly distributed? Is it strong evidence that there is some meaningful mechanical process at play, like a router that routes exactly every 60 seconds on the dot?

First, it's easy to compute that under the null hypothesis, the expected number of trades with a "0" for the seconds value of a given timestamp is $\frac{1000}{60} \approx 17$. So 30 is definitely much higher than expected, but what's the probability of a result like this? To figure this out, we could do an extended mathematical exercise. We could define random variables $X_1, \dots, X_n$ for $n = 1000$ to be identically distributed indicator variables, where X_i equals 1 if the i^{th} trade has a time stamp with seconds value equal to 0, and X_i equals 0 otherwise. In this case, we can define $X = \sum_{i=1}^n X_i$, and we're interested in the probability that $X \geq 30$. Variables X_i of this form are called Bernoulli trials, and the distribution of X is known as the binomial distribution. We can write a precise expression for the probability that X takes each possible value from 0 to n :

$$\mathbb{P}[X = k] = \binom{n}{k} p^k (1 - p)^{n-k},$$

where p is the probability that each $X_i = 1$. In our case, $p = \frac{1}{60}$ (under the null hypothesis).

We could then write a mathematical expression for the probability that $X \geq 30$, plugging in $n = 1000$ and $p = \frac{1}{60}$:

$$\mathbb{P}[X \geq 30] = \sum_{k \geq 30} \mathbb{P}[X = k] = \sum_{k \geq 30} \binom{1000}{k} \left(\frac{1}{60}\right)^k \left(\frac{59}{60}\right)^{1000-k}.$$

We could plug this into specialized math software like Mathematica to compute, but we can also start to see here that as we wade further into complicated scenarios, this exact mathematical approach may quickly become unwieldy, and simply too much of a pain to compute precisely.

So let's skip the math! (Shh... just don't tell anyone, ok? We spent way too much time learning math to admit it isn't always the best approach.) There is a simple advantage we have here that we have neglected: sampling from the distribution of X is easy. We simply flip a biased coin 1000 times, and count how many times we get HEADS. Once we've sampled X , checking whether the value is ≥ 30 is also easy. Doing this once (sampling X and checking

our criterion) is one *Monte Carlo simulation*. If we do it a bunch of times, we can get a pretty decent estimation of the probability the X satisfies our criterion. Here is some code that does this in python:

```
11 from numpy.random import randint
12
1 numExperiments= 1000000
2 m=0
3 for i in range(numExperiments):
4     r = randint(0,60,1000) #sample 1000 numbers randomly from the range from 0 to 59 (inclusive)
5     X = np.bincount(r)[0] #define X as the number of times the value 0 was chosen
6     if (X>=30):
7         m = m+1
8 print (m/numExperiments)
0.00179
```

Here we are doing 1 million simulations, and our estimate for $\mathbb{P}[X \geq 30]$ is less than 0.2%. If we run this code again, we'll get a slightly different estimate (due to the fresh randomness selected in the simulations), but the qualitative answer won't change really. It is clear that 30 samples out of 1000 having the same second value is not something that we would expect to see with a decent probability if all of the timestamp second values were independent and uniformly distributed.

We might want to generalize our question a bit here: what are the odds that *some* seconds value would occur at least 30 times? Presumably the only reason we focused in on 0 was because we looked at our data and found 0 to be a weirdly common value. If we had found 30 instances where the seconds value was 28, would that have been equally suspicious? Yes! So a better question here is actually: what's the probability that the mode of the samples occurs ≥ 30 times? Here's some python code to estimate this with Monte Carlo simulations too:

```
1 numExperiments= 1000000
2 m=0
3 for i in range(numExperiments):
4     r = randint(0,60,1000) #sample 1000 numbers randomly from the range from 0 to 59 (inclusive)
5     X = np.bincount(r).max() #define X as the number of times the mode occurred
6     if (X>=30):
7         m = m+1
8 print (m/numExperiments)
0.107475
```

As we can see, this probability is much higher, around 10%. This demonstrates that allowing the data itself to influence what analyses we perform can drastically skew our interpretation of the results. This is an insidious source of bias that we need to be constantly wary of. [Note: this analysis of the mode should remind you of our earlier discussion of collisions as way of detecting deviation from a uniform distribution. Often the same result/approach can be derived from different perspectives.]

Stepping back from this example, what we have here is a very general tool for estimating probabilities whenever we have an efficient sampling procedure. This can be an approximate replacement for complicated exact math, or even work in cases when we don't know how to express our probabilities in a clean mathematical formula. This often frees us to do hypothesis testing without needing to assume that distributions are mathematically nice (e.g. Gaussian).

7.3 Random Walks, and the Downside of Acting on Questionable Hypothesis Testing Results

An appealing application of hypothesis testing to finance is A-B testing to compare potential trading algo designs. Often, we would want to compare the current design (let's call it A) to proposed new design (let's call it B). The null hypothesis might be that the performance of the algos is the same on whatever particular benchmark we're interested in. For concreteness, let's say we're interested in slippage vs. arrival in terms of bps, in line with the example we considered in a previous section. To test our hypothesis, we might randomly assign orders to be traded by algo A or algo B, hoping that any important differences in order characteristics will balance out over the length of our experiment, so that the algos are ultimately trading somewhat comparable sets of orders in our data set.

By the end of our experiment, our data may look something like this (extending our prior example and showing only the most relevant columns to this discussion):

Algo	Side	Size	NV	...	SlippageVsArrival
A	Buy	20000	\$ 2004680	...	25 bps
B	Sell	1246	\$ 32445.84	...	-8 bps
B	Sell	156000	\$ 9817860	...	12 bps
A	Buy	5000	\$ 54550	...	-371 bps
B	Sell	7290	\$ 1792188.18	...	7 bps
B	Buy	2345	\$400479.10	...	-32 bps
A	Sell	499000	\$10588780	...	5 bps
A	Sell	7492	\$376547.92	...	18 bps
⋮	⋮	⋮	⋮	⋮	⋮

Ideally, we would collect *a lot* of data, and would include additional order characteristics, like what percentage of the symbol's ADV this order size represents. If we are super lucky (read: A DATA SCIENCE MIRACLE!), the distribution of the orders traded by algorithm A and the distribution of orders traded by algorithm B will be roughly the same when we

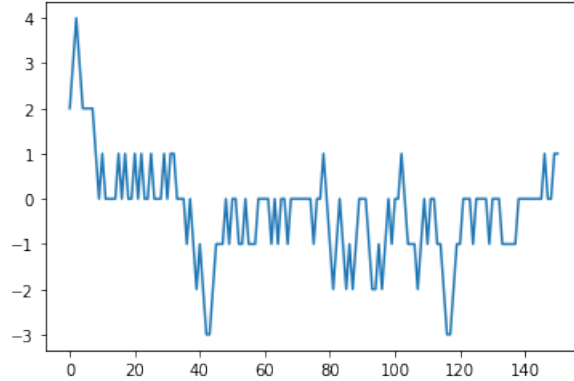
look at the most relevant order characteristics or combinations thereof.

In this lucky world where the order distributions are similar, we want to test the hypothesis that the algorithm's average performances are the same. Let's say we compute the notional value-weighted slippage vs. arrival in bps for the orders traded by algorithm A, and the value is 16 bps. We do the same for the orders traded by algorithm B, and the value is 13 bps. What is the probability of a result like this in a world where the performance of the algorithms are actually the same in expectation? Well, it depends on how we think about performance for each algorithm as a probability distribution. A basic approach would be to treat each order's slippage as an independent sample from a Gaussian distribution (either Gaussian A or Gaussian B). The weighted average of Gaussian samples is distributed as a Gaussian, so we can easily compute the probability of a difference ≥ 3 bps.

If we don't think the Gaussian model is a good model of our performance distributions, we have a range of options. We could use the overall empirical distribution we observe across A and B orders, for example, and re-sample outcomes from this using Monte Carlo simulations to estimate the probability of getting a difference ≥ 3 bps in the two averages simply by chance, under the hypothesis that both A and B distributions are equal to the overall empirical distribution. Beyond a Gaussian or empirical approach, there is also a wide range of models we could fit to our distributions, and this is a topic we'll return to later. But for now let's just suppose ... drum roll please ... that the 3 bps difference has a pretty substantial probability of happening by chance. For concreteness, let's say we estimate a 40% chance of a difference ≥ 3 bps occurring by chance.

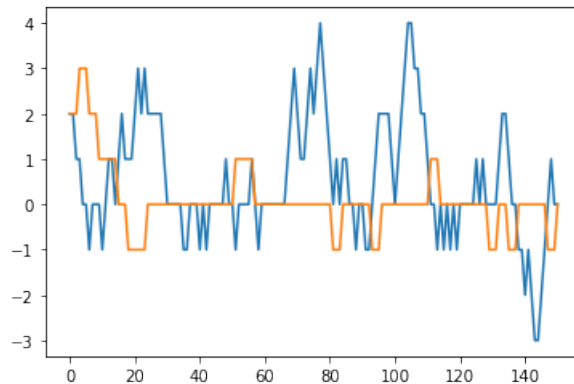
So what should we do? Should we change to algo B since it "won" the A-B test, even though the results were not particularly conclusive? What happens if we sequentially make a lot of decisions like this, making changes to the algo with somewhat weak evidence supporting each change?

Here's a visualization of what the end results of a process like this might be. This is over-simplified and fake, of course, but should provide a general sense of the phenomena at play. Suppose that an algo has one parameter, x , which can be set to any integer value. It is initially set to a value of 2, very close to the optimal value of 0. At each time step, we'll perform a test on three different versions of our algo: one with the current value of x , one with a value of x increased by +1, and one with a value of x decreased by -1. In each case, we'll correctly conclude that the value of x closest to 0 is better with probability 60%, while incorrectly concluding that one of the other two values is better with probability 40%. (We'll assume a 20% chance for each of the other values winning the test.) Let's simulate what happens to x as a function of time:



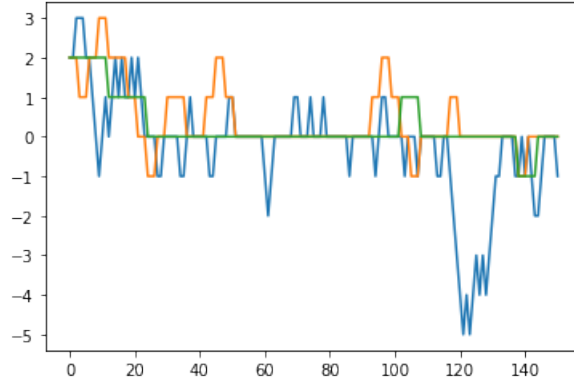
This is a random simulation over 150 experiment steps. This could be many more steps than the number of tests a single algo parameter is subjected too, but it's a long enough simulation so that we can get a general sense of the behavior. There are a few things we should notice here. First, since we are open to moving in each step, we fairly quickly arrive at the optimal value of 0, though we do happen to take a wrong turn initially in this particular simulation. However, this optimal value of 0 isn't very stable, as we are quite willing to move away from it again on weak evidence. This leads us to oscillate around 0.

Let's compare this to what happens if we make slower movements with higher quality evidence. In this case, we'll also simulate a process that only performs a test every three steps, and has a probability of 0.25 of making a sub-optimal move, rather than 0.40. Our quicker, high error simulation is performed in parallel and still graphed in blue, while our slower moving process is graphed in orange:



In this simulation, the orange process takes longer to arrive at the optimal value of 0 for the first time, but once it does it is more stable, deviating a bit less on average away from the optimal value than the more fickle blue process.

Taking this to a further extreme, we'll simulate again and add a green process that only performs an experiment every 6 steps, and makes a sub-optimal move with probability 0.1:



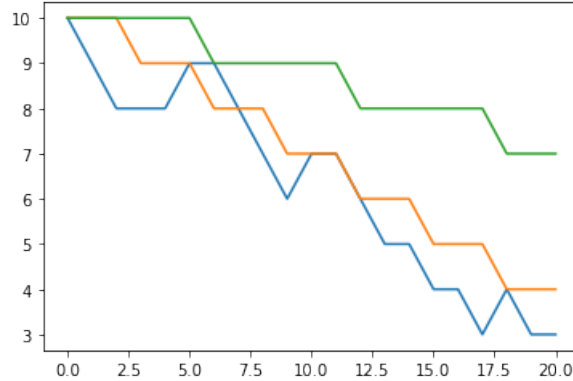
Again we can see that the sleepier and steadier processes “wobble” less away from the optimum once they arrive at it, and this results in less distance from the optimal value overall over the course of the simulation.

We can quantify this phenomenon by averaging over many simulations. For each simulation, we’ll compute the average distance from the optimal value of 0 over the course of the simulation, starting each simulation at the value 2. Then we’ll average these averages over many simulations. We’ll run 1000 simulations with 150 steps each.

Here are our averaged results, for the different experimental error probabilities and different number of steps per experiment that we have been comparing:

Error Probability	Steps Per Experiment	Avg distance from optimum
0.4	1	0.77
0.25	3	0.43
0.1	6	0.24

Naturally, these numbers will shift if we shorten our simulation length, or change our starting position. If we had a small number of simulation steps and a starting position very far from 0, we would expect the faster stepping curves to outperform the slower moving ones, just because they would be taking faster steps (mostly) in the right direction. Here is a simulation of 20 steps with a starting position of 10, where the blue curve again represents experiments with error probability of 0.4 performed every step, the orange curve represents experiments with error probability 0.25 performed every 3 steps, and the green curve represents experiments with error probability of 0.1 performed every step:



This makes intuitive sense: if we are far away from the optimal value, we are better off stepping quickly, even if each step is less confident. But if we are near the optimal value, our willingness to act on weak signals will cause wider oscillation around the optimal value and this will make a meaningful dent in performance. The trick of course, is that we won't really know when we are close to the optimal value and when we are far. There are things we could try though, like looking at our recent history of experiments to try to detect if we are oscillating around a value or largely moving in one direction. In this scenario, we may want to dynamically set our tradeoff between speed and accuracy of experiments.

Regardless, the core lesson here is that being willing to act quickly on weak but positive signals is good in some circumstances, but certainly not optimal in all. A more mature approach to iterative experiments should consider more context in deciding when to act, rather than always following the winner of an A-B test, especially if the statistical power of the conclusion is weak.

Chapter 8

Correlation Testing

There are naturally a lot of times in analyzing financial data when we ask: “do these two things have anything to do with each other?” For example, does the size of an order have anything to do with its performance? Does which broker handled an order have anything to do with its slippage vs. arrival?

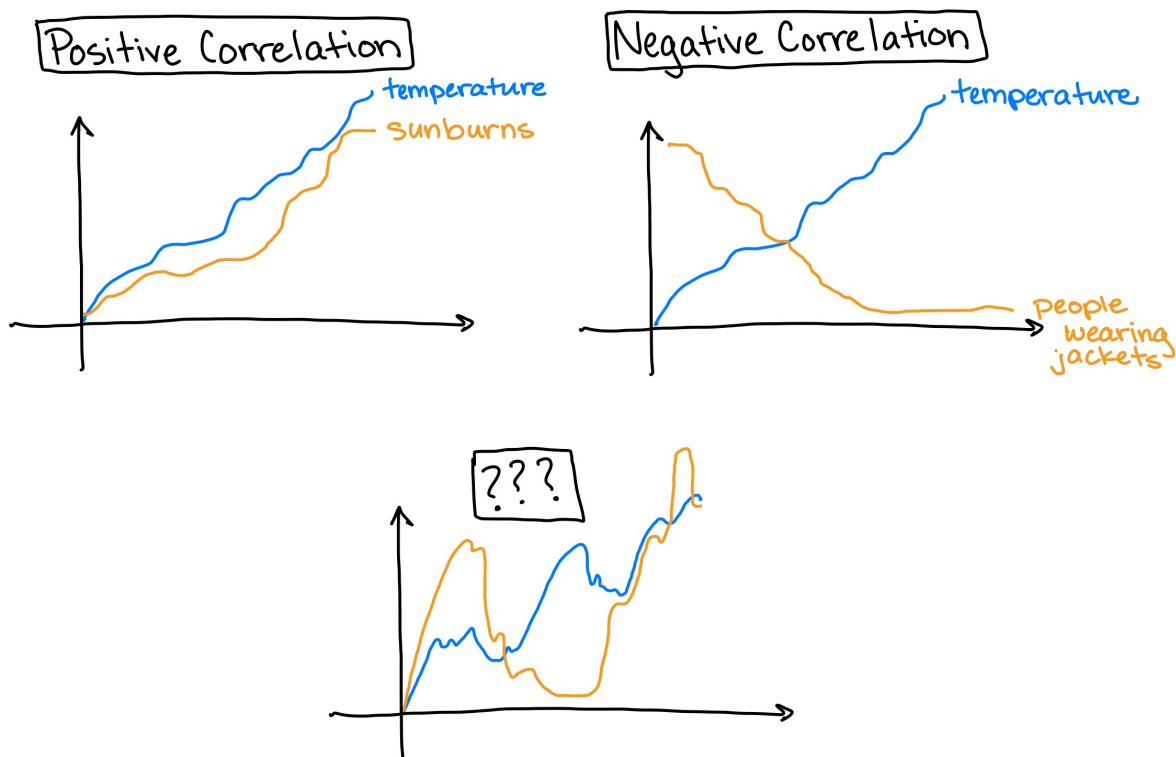
It can be tempting to jump right into building models as a first approach to these questions. Well, let’s find a best fit line between order size and order performance, we might say, and see how positive or negative the slope is! But there’s a reason this chapter comes before the later chapter on model building. Asking a model if variables have a relationship is like asking a lawyer if you need legal services. They are always going to say yes - it’s what they do! The trouble is, the “best” model might not be good. Goodness-of-fit tests can show you this of course, but sometimes there no substitute for going back to the basics instead of racing ahead to model building. The space of models is big, as is the space of features that can go into models. Building a little intuition first by answering more basic questions on your data and can save you a lot of time in chasing model shapes/sizes/features that are not a good fit (pun-intended) for your data.

So - what does it mean, at the most basic level, for two things to be related? It depends on what kind of “things” we’re talking about. We’ll think about this question for three different cases: 1) numerical things, 2) categorical things, 3) mixes of numerical and categorical things.

8.1 Numerical Variables

For numerical things, two things being “related” usually means that they tend to go up or down together, or they tend to go in opposite directions. When we qualitatively think about “positive correlation,” we probably picture an upward trending curve. Average temperature goes up, number of sunburns goes up. When we qualitatively think about “negative cor-

relation,” we probably picture a downward trending curve. Average temperature goes up, number of people wearing winter coats goes down.



Real data tends to be noisy (especially in finance), so the trends might be weak and hard to see with the naked eye. So how do we mathematically define and measure something like the tendency of two numerical quantities to go up/down together? One way is the *Pearson correlation coefficient* [16].

To explain the reasoning behind the formula for this, let's start with an example. Let's suppose that two friends, Alice and Beth, go gambling at a local casino together many times. On average, each of them ends up losing about as much as they win, and their total winnings/losses are flat by the end of the trips. They might wonder - did their "lucky" nights tend to coincide? If so, this could be a reflection of the skill of dealers and other players that they encountered. Alice has a record of her winnings and loses per evening, which we'll denote by $A_1, \dots, A_n$. Beth similarly has a record of her wins and loses, which we'll denote by $B_1, \dots, B_n$. Let's think about the sequence of products, $A_1B_1, \dots, A_nB_n$. On a night where Alice and Beth both won money, the product will be positive. On a night where Alice and Beth both lost money, the product will also be positive. On nights where one of them won and one of them lost, the product will be negative. So we might expect that the average of these products will tell us something about whether Alice and Beth experienced correlated outcomes.

Just to make this super concrete, let's suppose that $n = 10$ and the values are:

$$(A_1, \dots, A_{10}) = (+200, +100, +100, -200, +0, -500, +100, +0, +100, +100)$$

$$(B_1, \dots, B_{10}) = (+300, -100, +100, -300, -200, -400, +200, +100, +400, -100)$$

Notice here that $\sum_i A_i = 0 = \sum_i B_i$. Now let's compute and sum up the products:

$$(A_1 B_1, \dots, A_{10} B_{10}) = (+60000 - 10000 + 10000 + 60000 + 200000 + 20000 + 40000 - 10000)$$

$$\sum_i A_i B_i = 350000,$$

yielding an average product of $350,000/10 = 35,000$. Intuitively, there seems to be a strong positive correlation here.

But our simplistic example here could get more complicated in several ways. Let's imagine, for example, that Alice is a very poor gambler, and Beth is a very skilled one. In this scenario, perhaps their values look more like this:

$$(A_1, \dots, A_{10}) = (-100, -200, -200, -500, -300, -800, -200, -300, -200, -200)$$

$$(B_1, \dots, B_{10}) = (+700, +300, +500, +100, +200, +0, +600, +500, +800, +300)$$

It's still true here that the nights where Alice does a little less poorly tend to coincide with the nights where Beth does a little better. But now the average product of their wins/losses is going to be extremely negative, since every product here is negative in sign. What's happening here is that the fixed skill deficit between Alice and Beth is driving the product negative, obscuring the positive correlation effect. To get back to a clean view of the correlation, we need to re-center Alice's values and Beth's values around 0. [Aha! There's that data normalization concept coming up again!] You might notice that the sum of Alice's values is now -3000, and if we add +300 to each value, we get back to the original example. Similarly, the sum of Beth's values is now +4000 and subtracting -400 from each value gets us back to the original example.

We might also wonder: what happens if Beth is just a much wilder gambler than Alice, playing with higher sums and taking more risks. This will likely lead to a higher variance in her outcomes. We could imagine this as a multiplier on Beth's values: perhaps all of her values get multiplied by 10, say. This would clearly multiply the average product of Alice and Beth's values by the same amount. While this wouldn't change the sign of the product, it would obscure the meaning of its magnitude. If the average product is intended to be

a measure of correlation, then it isn't very useful if it can be arbitrarily scaled by phenomena that are separate from the degree of correlation. We demonstrate this phenomenon visually in Figure 8.1 for the three above examples.

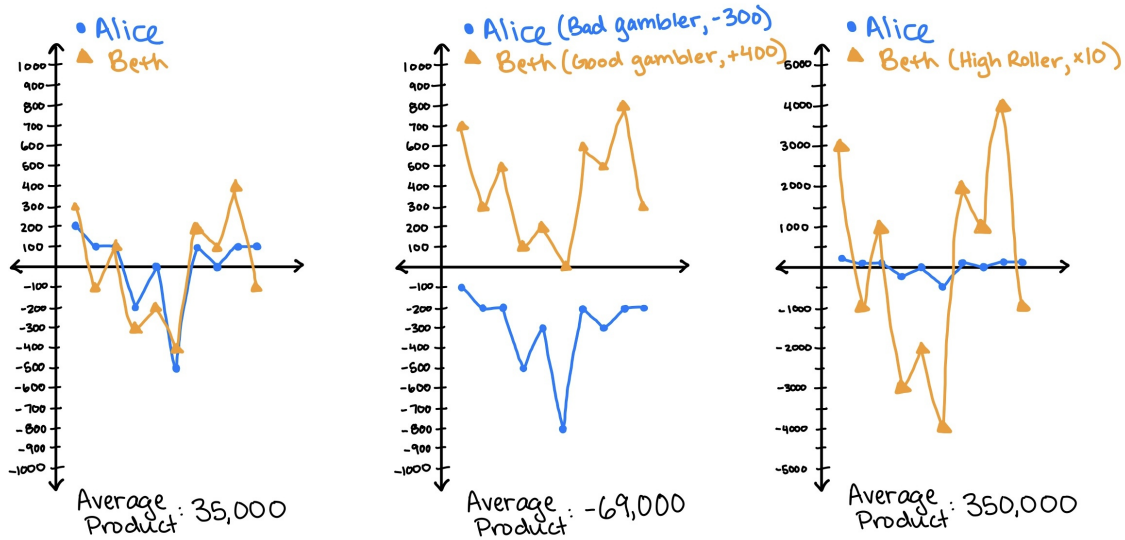


Figure 8.1: Raw average product is not an ideal metric for determining correlation between two data sets, because shifting or amplifying the data points does not affect the amount to which they move together directionally, but does affect the average product significantly in value and sign.

To correct for this, we can re-scale Alice and Beth's values individually to have variance equal to 1. These re-centering and re-scaling corrections yield the formula for the Pearson correlation coefficient, which we'll express here for two random variables X and Y :

$$\mathbb{E} \left[\frac{(X - \mu_X)(Y - \mu_Y)}{\sigma_X \sigma_Y} \right].$$

Here, μ_X denotes the expectation of X and μ_Y denotes the expectation of Y . Similarly, σ_X^2 denotes the variance of X and σ_Y^2 denotes the variance of Y . Notice that we're not really comparing X and Y directly here - we're comparing the re-centered and re-scaled versions of X and Y . The variable $\tilde{X} := \frac{X - \mu_X}{\sigma_X}$, for instance, is an adjusted version of X that has mean 0 and variance 1. Doing this adjustment to each of X and Y individually ensures that we are making a more apples-to-apples comparison, and that the magnitude of our measurement is not being inflated/deflated due to the relative scales of X and Y .

8.1.1 Investigating the Pearson Coefficient

As with any new formula, it can be helpful to build up our intuition by considering some edge cases. We might ask: what's the largest (positive) value that a Pearson coefficient can take? Because of the re-centering and re-scaling, the largest possible value is 1. Let's work out why this is true. One quick way is to apply the Cauchy-Schwarz inequality, but we know very few people who walk around with an internalized understanding of why the Cauchy-Schwarz inequality is true, so this is probably not super helpful. Instead, let's simplify our notation a bit by defining $\tilde{X} = X - \mu_X$ and $\tilde{Y} = Y - \mu_Y$ and then rewriting:

$$\mathbb{E} \left[\frac{(X - \mu_X)(Y - \mu_Y)}{\sigma_X \sigma_Y} \right] \leq 1 \iff \mathbb{E}[\tilde{X}\tilde{Y}] \leq \sqrt{\mathbb{E}[\tilde{X}^2]\mathbb{E}[\tilde{Y}^2]}.$$

Hence it suffices to show that

$$(\mathbb{E}[\tilde{X}\tilde{Y}])^2 \leq \mathbb{E}[\tilde{X}^2]\mathbb{E}[\tilde{Y}^2].$$

There are many ways to prove this statement, and many that make no assumptions about the discreteness of the probability space. However, we usually find it most enlightening to work in a setting where the probability space is finite, and to write out all of these things in terms of sums. If we let $\{x_i\}$ denote the possible outcomes of $\tilde{X}$ and $\{y_j\}$ denote the possible outcomes of $\tilde{Y}$, we can get underneath the elegant notation of expectations and look at our desired inequality in terms of raw sums:

$$\sum_{i,j,k,\ell} x_i y_j x_k y_\ell p(x_i, y_j) p(x_k, y_\ell) \leq \sum_{i,\ell} x_i^2 y_\ell^2 p(x_i) p(y_\ell).$$

The indices i and k here range over the possible outcomes of $\tilde{X}$, while the indices j and ℓ range over the possible outcomes of $\tilde{Y}$. The values $p(x_i, y_j)$ denote probabilities over the joint space for $\tilde{X}$ and $\tilde{Y}$, while the values $p(x_i)$ denote probabilities over the space for $\tilde{X}$, etc.

It can be easy to get lost when staring at a mess like this. But we have chosen the indices carefully to provide a clue for how to proceed. We might guess that it would help, for example, to make the sides of the inequality look more similar. For this, we can translate the single probabilities $p(x_i)$ and $p(y_\ell)$ on the righthand side into expressions involving joint probabilities:

$$p(x_i) = \sum_j p(x_i, y_j), \quad p(y_\ell) = \sum_k p(x_k, y_\ell).$$

Plugging these in, our desired inequality becomes:

$$\sum_{i,j,k,\ell} x_i y_j x_k y_\ell p(x_i, y_j) p(x_k, y_\ell) \leq \sum_{i,j,k,\ell} x_i^2 y_\ell^2 p(x_i, y_j) p(x_k, y_\ell).$$

Now we can see some common structure, and combine these terms into one sum over our four indices:

$$\sum_{i,j,k,\ell} (x_i y_j x_k y_\ell - x_i^2 y_\ell^2) p(x_i, y_j) p(x_k, y_\ell) \leq 0.$$

Staring at this, we might wonder - is each individual term here ≤ 0 ? That would make the inequality true in the most obvious way. Since they are probabilities, the terms $p(x_i, y_j)$ and $p(x_k, y_\ell)$ are always non-negative, so this is equivalent to asking if $x_i y_j x_k y_\ell - x_i^2 y_\ell^2$ is always ≤ 0 . This isn't always true though - suppose for example that $y_j = y_\ell$, but $x_k > x_i > 0$. Then this term will be positive. We might notice, though, that in this case the term will become negative when the roles of indices i and k are flipped. More generally, for every set of values for (i, j, k, ℓ) , there is complimentary set where i and k are exchanged, and j and ℓ are exchanged. The sum of these two terms taken together will be (written in the original indices):

$$2x_i y_j x_k y_\ell - x_i^2 y_\ell^2 - x_k^2 y_j^2 = -(x_i y_\ell - x_k y_j)^2.$$

Aha! now we can see that this will be ≤ 0 . Since the overall sum over values for (i, j, k, ℓ) can be decomposed into a sum over pairs of such complimentary values, the entire sum is a linear combination of squares with all negative coefficients, and hence is ≤ 0 , as we needed to prove. Similarly, the lowest value that the Pearson coefficient can take is -1 .

Now that we know the range of values is limited to -1 to 1 , we probably want to know: so how far away from 0 should a value be in order to indicate a meaningful correlation? Unfortunately, this isn't something that we can simplify with a standard definition. The answer is going to be context dependent. One important factor to keep in mind here is that when we compute Pearson coefficients on real data, we're dealing with samples of distributions, not the distributions themselves. And so the level of precision and confidence in our estimate of the Pearson coefficient should play a role in our interpretation of the value.

One general way to get a handle on the potential error in our estimate is to do Monte Carlo experiments, as discussed in the previous chapter. We could re-sample independent variables, for example, from the same empirical distributions we observe for X and Y , and look at the Pearson coefficients we get on the resulting simulated data sets. If the real value we computed is well inside this range, we should view it as consistent with a world where the variables are uncorrelated.

It is also important to remember that the Pearson coefficient is simply a weighted average of products, and hence is subject to the common pitfalls of weighted averages. Namely, it can be heavily influenced by outliers. In our example, if Alice or Beth had just one *extreme* result, this single number could dominate the result and obscure any common pattern in the more modest numbers.

8.1.2 Investigating the Spearman Coefficient

Because of this non-robustness of the Pearson coefficient, it can help to consider a variation known as the Spearman coefficient [19]. To compute a Spearman coefficient from two sets of samples for two random variables, we replace the actual values with their ranks. Let's return to our prior example of Alice and Beth's gambling results, and imagine that there are now a few outliers (we'll also make the values all distinct to make the ranks clear):

$$(A_1, \dots, A_{10}) = (-100, -200, -250, +100, -300, -10000, +200, +300, +2000, -400)$$

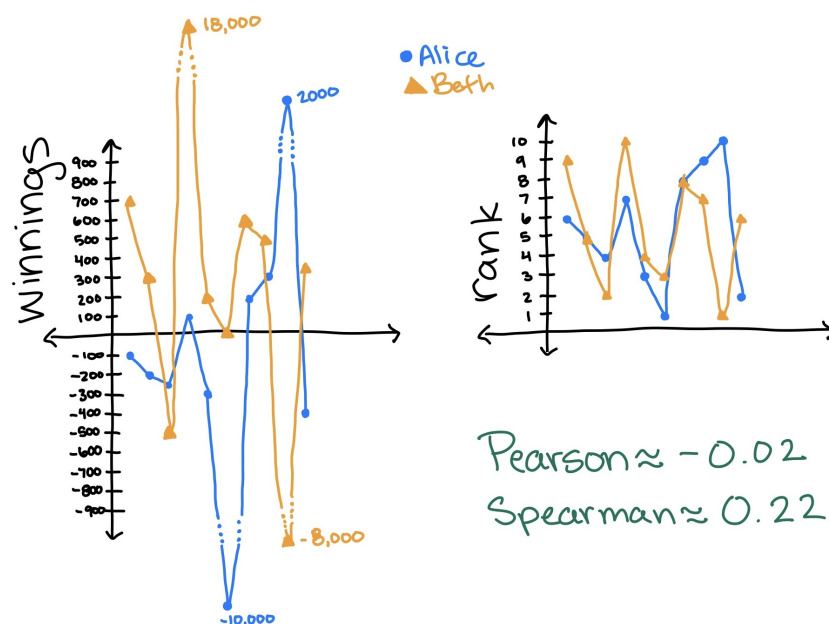
$$(B_1, \dots, B_{10}) = (+700, +300, -500, +18000, +200, +0, +600, +500, -8000, +350)$$

Denoting the ranks by A'_i and B'_i in order from most negative to most positive, we have:

$$(A'_1, \dots, A'_{10}) = (6, 5, 4, 7, 3, 1, 8, 9, 10, 2)$$

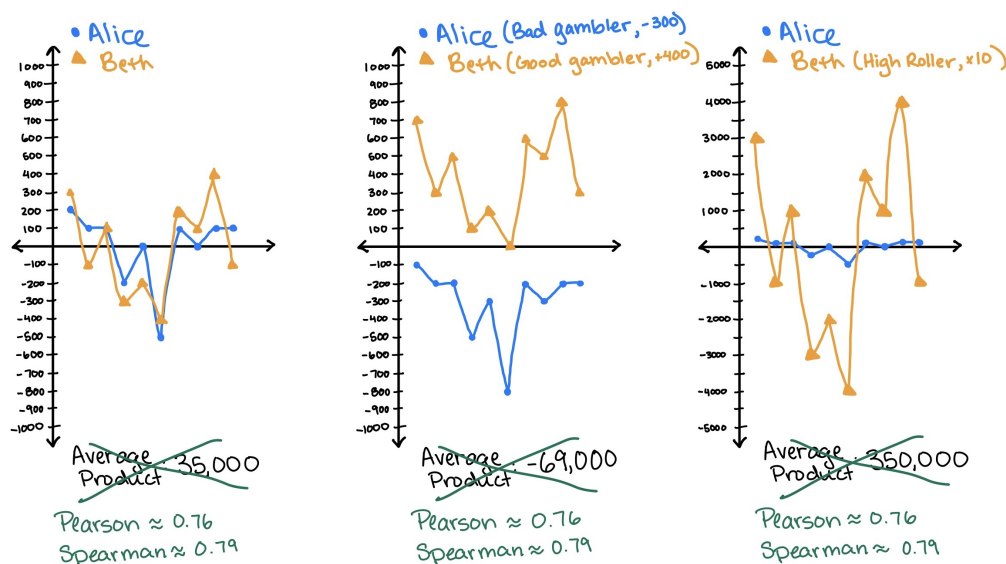
$$(B'_1, \dots, B'_{10}) = (9, 5, 2, 10, 4, 3, 8, 7, 1, 6)$$

Expressed visually, this is:



Let's take a moment to note what we have lost and gained by doing this. We've lost information about how wildly disparate the underlying values might be - it's not possible to know from these ranks, for instance, how much money was won or lost, or who ended up with higher average winnings. What we've gained is a representation of our data that is relatively unperturbed by singular events, and also unaffected by the particular shape of an upward or downward trend. This is a good basis for finding correlations that might not be linear, as long as they are relatively monotonic. After performing this translation, the Spearman coefficient is computed as the Pearson coefficient of the ranks. Since it is also a Pearson coefficient (just of the ranks, rather than the raw numbers), it is also guaranteed to fall inside the range from -1 to 1.

If we go back to our original examples of shifting and amplifying the values of winnings for Alice and Beth, we see that both the Pearson and Spearman coefficients are unaffected by these adjustments, and therefore capture the correlation information more robustly than the originally proposed product approach.



8.2 Categorical Variables

Sometimes we want to know if two things are related, and neither thing is a number. For instance, let's suppose we want to check if a particular broker is routing orders to buy stock differently than they are routing orders to sell stock. We'll start with an extremely basic example, where the broker routes to just two trading venues (exchanges) that we'll call A and B. Let's suppose we have a data set with a bunch of trades, and for each we have a field for side that either says "Buy" or "Sell," and a field for venue that either says "A" or

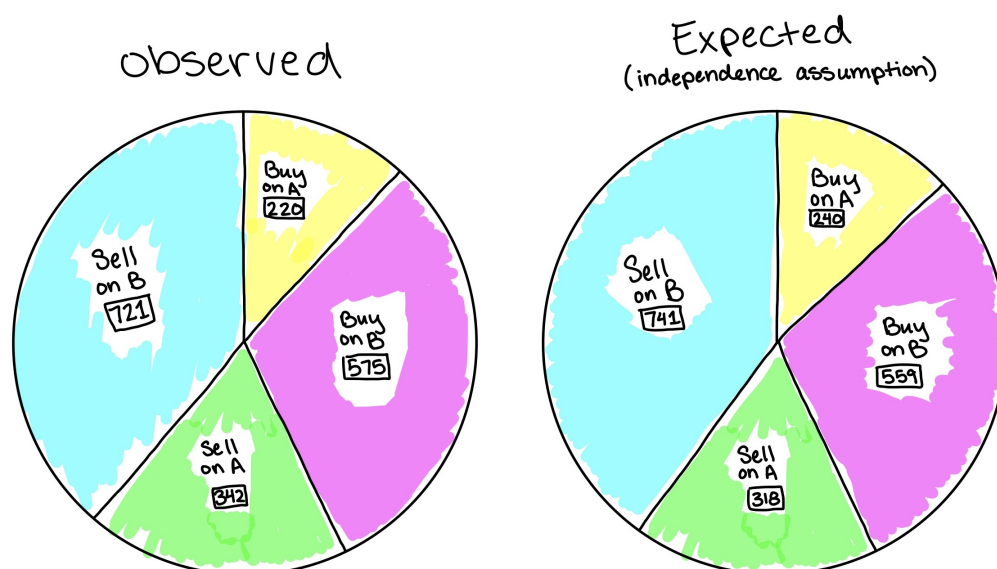
“B.” In total here, there are four possible combinations. Buy orders that traded on venue A, sell orders that traded on venue A, buy orders that traded on venue B, and sell orders that traded on venue B. We can count how many orders fall into each of the four categories and make a simple chart:

	A	B
Buy	220	575
Sell	342	721

Table 8.1: Counts of trades in each category

We can see a few trends here: there are more sells than buys, and also more trades on venue B than venue A. But it may not be obvious by eye if there’s a meaningful relationship between the venue split and the order side. We can start analyzing this by asking: what counts would we have expected if there were no relationship? In total here, there are 1858 trades. Roughly 43% of them are for buy orders, while 57% are for sell orders. Roughly 30% of them occur on venue A, while 70% occur on venue B.

If the side of the order and the venue of the trade were independent random variables, we would then expect that the fraction of orders falling into the “Buy on venue A” category would be $0.43 * 0.30 = 0.129$, so this would be $0.129 * 1858 \approx 240$. The expected fraction of orders falling into the “Buy on venue B” category would be $0.43 * 0.70 = 0.301$, representing about 559 orders. The expected fraction of orders falling into the “Sell on venue A” category would be $0.57 * 0.30 = 0.171$, representing about 318 orders. The expected fraction of orders falling into the “Sell on venue B” category would be $0.57 * 0.70 = 0.399$, representing about 741 orders.



Now the numbers we observed did not match these expected numbers exactly - but are they far enough away from them to indicate something meaningful is skewing the results, or are they reasonably consistent with purely random deviation?

This question can be answered using the chi-squared test. Let's let m_i denote the expected count for category i under the hypothesis that the variables are independent, and x_i denote the actual count. Then we can compute:

$$\chi^2 := \sum_{i=1}^4 \frac{(x_i - m_i)^2}{m_i}.$$

This formula gives us a quantity called *the χ^2 statistic*.

The distribution of this that emerges in the limit as the sample size gets larger is known as the χ^2 distribution, and it is parameterized by the *degrees of freedom*. In general, if we have two categorical variables that take on r and c different values respectively, the degrees of freedom can be calculated as $(r-1)(c-1)$. In our example, each variable takes two possible values, so we have just 1 degree of freedom.

The χ^2 distribution with one degree of freedom can be generated by taking a normal Gaussian (mean 0, standard deviation 1) and squaring it. Naturally, this becomes a distribution over non-negative numbers, which squares (get it?) with the fact that our formula for the χ^2 statistic can only produce non-negative values. More generally, the χ^2 squared distribution with k degrees of freedom can be produced by summing k squares of normal Gaussian distributions.

8.2.1 Investigating the Chi-Squared Test

To gain a better intuition for what's going on here, let's dig a little deeper into why the distribution of the χ^2 statistic for our example converges to the square of a normal Gaussian distribution as the sample size increases. Let's let N denote our total number of data points. We'll let S_i be a random variable that equals 0 if the side of the i^{th} data point is "Buy," and equals 1 if it is "Sell." We'll let V_i be a random variable that equals 0 if the venue of the i^{th} data point is A and equals 1 if it is B . We define:

$$S := \sum_i^N S_i, \quad V := \sum_i^n V_i,$$

so S is the total number of sell orders and V is the total number of trades on Venue B . We'll let q denote the fraction of sell orders, meaning that $S = qN$, and p denote the fraction of trades on Venue B , meaning that $V = pN$. To see why this situation has one degree of

freedom, let's also give a name C to the number of trades on Venue B that correspond to sell orders (i.e. the number of indices i such that $S_i = V_i = 1$).

Now we can fill in our chart above in terms of these variables, where every other entry is determined once we have named C and specified p , q , and N :

	A	B
Buy	$N - qN - pN + C$	$pN - C$
Sell	$qN - C$	C

Table 8.2: Counts of trades in each category

Now, let's imagine (falsely, but only somewhat), that p and q are not empirically derived from our data points, but instead represent the ground truth expectations of random variables $\{V_i\}$ and $\{S_i\}$ respectively, and that these are sampled independently from each other and independently over i . In such a world, the expected values for each cell in our table would be:

	A	B
Buy	$(1-q)(1-p)N$	$p(1-q)N$
Sell	$q(1-p)N$	pqN

Table 8.3: Counts of trades in each category

So our χ^2 statistic computation would then become:

$$\begin{aligned} \chi^2 = & \frac{(N - qN - pN + C - (1-p)(1-q)N)^2}{(1-p)(1-q)N} + \frac{(pN - C - p(1-q)N)^2}{p(1-q)N} \\ & + \frac{(qN - C - (1-p)qN)^2}{(1-p)qN} + \frac{(C - pqN)^2}{pqN}. \end{aligned}$$

We then observe that each of these numerators is equal to $(C - pqN)^2$. Don't believe us? Try:

$$\begin{aligned} N - qN - pN + C - (1-p)(1-q)N &= N - qN - pN + C - (1-p-q+pq)N \\ &= N - qN - pN + C - N + pN + qN - pqN \\ &= C - pqN. \end{aligned}$$

Still skeptical? Try:

$$pN - C - p(1-q)N = pN - C - pN + pqN = -(C - pqN).$$

We could do the last one, but we'll assume you trust us now.

Putting this all together over a common denominator of $pq(1-p)(1-q)N$ and noting that $pq + (1-p)q + (1-q)p + (1-p)(1-q) = 1$ (since these represent comprehensive and mutually exclusive probabilities for the joint values of V_i and S_i , we have:

$$\chi^2 = \frac{(C - pqN)^2}{pq(1-p)(1-q)N}.$$

Giving C a name was useful to simplify things to this point, but now we will substitute a different convenient notation. Let's observe that:

$$C - pqN = \sum_{i=1}^{i=N} (V_i - p)(S_i - q).$$

To verify this, recall that $C := \sum_i V_i S_i$, $\sum_i S_i = qN$, and $\sum_i V_i = pN$, so:

$$\sum_{i=1}^{i=N} (V_i - p)(S_i - q) = \sum_{i=1}^N V_i S_i - pS_i - qV_i + pq = C - pqN - pqN + pqN = C - pqN.$$

Let's look at the random variable $(V_i - p)(S_i - q)$ more closely. The expectation of this random variable is 0, since the expectation of $V_i = p$ and the expectation of $S_i = q$, and we are assuming (in this null hypothesis world) that the V_i and S_i are independent. As an additional consequence of the independence assumption,

$$Var[(V_i - p)(S_i - q)] = Var[V_i - p] * Var[S_i - q].$$

We can compute $Var[V_i - p]$, for example, as follows:

$$Var[V_i - p] = \mathbb{E}[(V_i - p)^2] = \mathbb{E}[V_i^2 - 2pV_i + p^2] = p - 2p^2 + p^2 = p - p^2 = p(1 - p).$$

Here we have used the fact that $V_i^2 = V_i$ because V_i is binary-valued, and hence $\mathbb{E}[V_i^2] = \mathbb{E}[V_i] = p$. Similarly, the variance of $S_i - q$ is $q(1 - q)$.

Putting this all together, we see that

$$\frac{\sum_{i=1}^N (V_i - p)(S_i - q)}{\sqrt{pq(1-p)(1-q)N}}$$

will converge to a normal Gaussian (mean 0, variance 1) as the value of N goes to infinity, by the Central Limit Theorem. And hence, the distribution of our χ^2 statistic will converge to the distribution of a random variable that is the square of a normalized Gaussian. If we

have a relatively large sample size, the gap between the real distribution and the limiting χ^2 distribution will not be very important. If we have a small sample size, it might be.

Let's suppose we feel our sample size is big enough to assume the limiting distribution, and return to our example. In our case, we have counts in each category equal to: $x_1 = 220$, $x_2 = 575$, $x_3 = 342$, $x_4 = 721$, while the expected counts are $m_1 = 240$, $m_2 = 559$, $m_3 = 318$, and $m_4 = 741$. Plugging these in, we get:

$$\chi^2 \approx 1.67 + 0.46 + 1.81 + 0.54 = 4.48.$$

We then ask: how likely is a value ≥ 4.48 under the χ^2 distribution with one degree of freedom? (i.e., the distribution of a normal Gaussian squared). If we plug this into an online calculator (or do a sufficiently large Monte Carlo simulation), we see that this probability is roughly 0.034. This means there is about a 3.4% chance of getting a χ^2 statistic at least this large in the null hypothesis world where there is no relationship between the trade venue and the order side. This is right on the border of interpretation: could be good evidence that there is a relationship here, but also could be due to chance. This is the kind of result that should prompt us to retest on fresh data if possible to shore up our evidence.

It should also be noted that there are a lot of things here we could have done differently. We could have avoided assuming the sample size was big enough to approximate the distribution of our χ^2 statistic by a squared normal Gaussian, for example, by doing Monte Carlo simulations at our true sample size instead. We could have tried to count how many shares corresponded to each category rather than how many orders/trades (though this could get quite tricky, as individual shares don't often trade independently). All of our little choices affect the outcome of the analysis, so it's very tempting to say: well, let's just try all of the reasonable variations and see if one of them gives an interesting result! This is very dangerous though, as we discussed previously. The more analyses we try, the higher the probability that we get a seemingly "interesting" result by chance.

8.3 Mixing Numerical and Categorical Variables

Sometimes we want to know if two things are related, where one thing is a number and the other is not. An example might be: does some performance metric (e.g. slippage vs. arrival) behave differently for sell orders versus buy orders? In this case, our question is a distribution testing question. Do the performance values for buy orders and the performance values for sell orders behave as if they are from the same distribution? Questions like these were discussed in a previous chapter already, so we will not rehash them here.

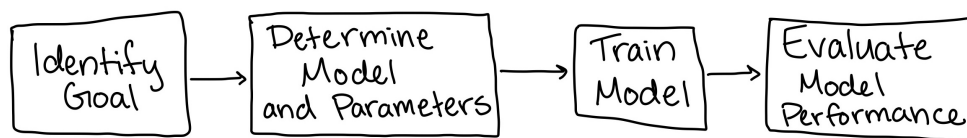
Chapter 9

An Overview of Machine Learning and Optimization Problems

Now that we have covered some of the specific nuances and pitfalls of working with trading data, as well as some important contextual tools like distribution testing and correlation coefficients, we are ready to discuss the more exciting task of building models from our data. This largely falls under the scope of “machine learning.”

Ever since “machine learning” and “artificial intelligence” escaped the containment of academic environments and become buzzwords in the popular lexicon, it’s become nearly impossible to sort out what people mean when they use them, and how these uses overlap with more basic-sounding words like “statistics” and “automation.” We cannot hope to sort that out here, but we can clearly define our own intended scope when we use the term “machine learning.” For our purposes, machine learning will refer to the collection of techniques used to extract structure from data.

There are a few common elements that are usually required to talk about machine learning precisely: (1) a model (mathematical object or collection of mathematical objects) defined over some set of parameters or weights, (2) a “training” or fitting procedure to determine the values of the model’s parameters and weights, and (3) evaluation or testing procedure to evaluate the quality of the fit of the trained parameters of the model. In general, there should be some clear separation between the data or methods used to train/fit the model and the data or methods used to test/evaluate the model. Not every machine learning technique we discuss will have an obvious evaluation procedure, however in all cases it is important to understand how you will evaluate a model before you begin training so as not to overfit the model to the exact scenarios you trained on.



It is important to note when discussing machine learning, that one model could be trained in various different ways, one training procedure could be used for many different models, and there are many different models and training procedures that can be used to accomplish the same goal or evaluation criteria. When looking at a machine learning technique or approaching a problem for which you want to use machine learning, it is important that you identify what is the goal and how to evaluate whether you have accomplished this goal, what kinds of models would make sense for achieving this goal, and what kinds of training procedures are both possible in your context and suitable for the models and goals you are considering.

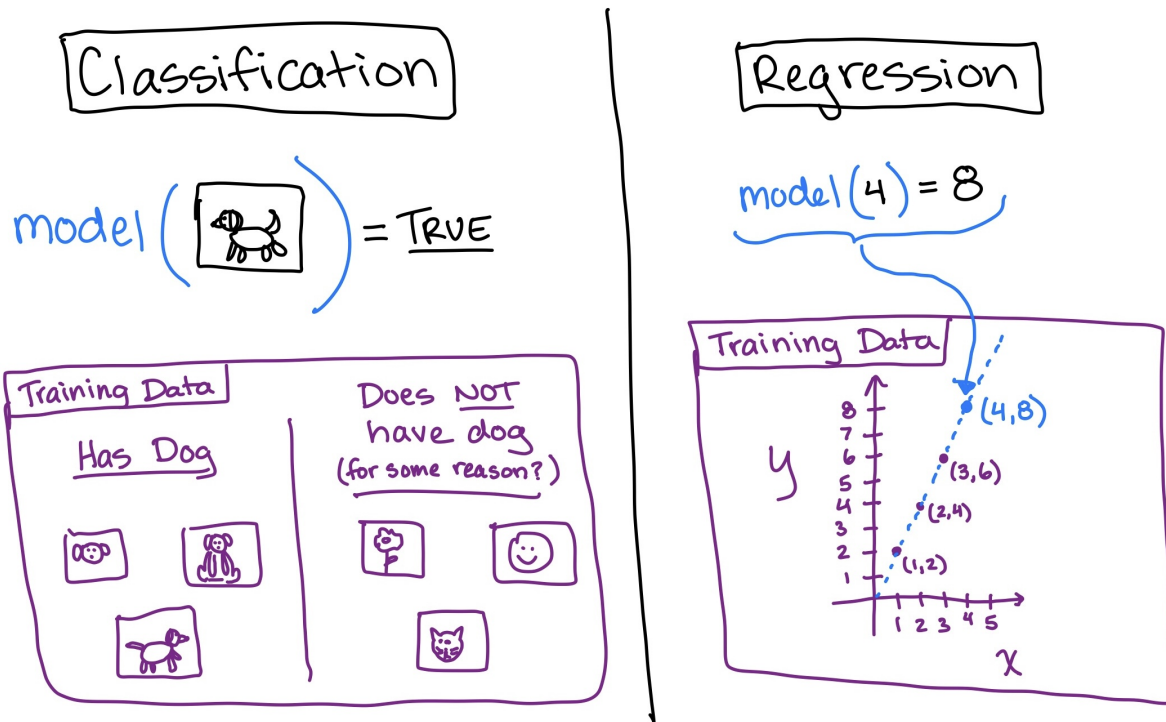
When categorizing the known machine learning techniques, we can broadly group them by two features: whether the training and testing data is available all at once or in a streaming fashion, and to what extent “answers” are provided. When training data is available at once, we refer to this setting as “offline” learning. When it is available in a streaming fashion, we called it “online.” When we have data to work with where the desired answers are known, we call this “supervised” learning. When we don’t get full answers, but we do get feedback on choices we make, we call this “reinforcement” learning. When there is no form of feedback, we call this “unsupervised” learning. These terms will hopefully make more sense after we have gone through some explanations and canonical examples.

9.1 Supervised Learning

Supervised learning is what one generally thinks of when discussing traditional machine learning, and it is often the first subject taught in a machine learning course. What makes it “supervised” is the access to a set of labels or “true” values about the data we are trying to learn. You can think of this like a spelling test in school. Your data source is the teacher reading out various strange words, and you (the machine learning model in this analogy) are trying to identify correct spellings. What makes this a supervised learning problem is that for each of the words, there is a known correct spelling, and you can be graded on your performance against a real answer key.

There are two main kinds of tasks that are solved via supervised learning: *classification* and *regression*. In classification, the machine learning model takes in a data point with

various features, and determines which label to predict for it from a pre-determined set of candidate labels. This is the classical “is there a dog in the photo?” case. In regression, the model takes as input a data point and outputs a predicted value from an infinite or continuous set of possible values. For example, if your training data is (x, y) pairs: $(1, 2), (2, 4), (3, 6)$, a good model for this data would be $f(x) = 2 * x$, meaning if you got a test input $x = 4$, you would output $y = 8$.



In most cases, the supervised setting is the most easily mapped to the modeling, training, evaluation paradigm of machine learning. Here, the model is a function that takes in a data point and outputs a prediction. The training procedure is some method for using the training data and true labels to update the parameter values of the model. Here, evaluation is particularly straightforward: you can determine easily how well your model performs by evaluating it over another set of “testing” data that is similar (but not identical) to the training data.

Supervised learning can be done in both an offline and online setting, and both are used widely. It is important to note the differences and understand when one setting might be more suitable than the other.

9.1.1 Offline Supervised Learning

Most of what we think of as classical statistics is performed on data that is available all at once, and for which the “right” answers are supposedly given. This is the setting of *offline supervised learning*. This encompasses scenarios like: you have collected a bunch of data from a clinical study. At the end of the study, you attempt to fit a model to your data to explain how some set of variables are related to the outcomes, like: how does age affect a treatment outcome? In this case, you have already collected all of the data and can refer to it all at once, and you have some supposed source of ground truth for the outcome you are attempting to model (you have recorded values for the treatment outcome for each case).

In our spelling test analogy, offline supervised learning is the case where you get a large set of words spelled correctly before the test and are able to study them, their roots, and how they are used in language. You can learn grammatical rules and how to conjugate verbs from an infinitive. When you go to take the test, you are likely given words you haven’t seen in your study materials, but you use what you learned about the language in general to identify what their most likely spelling is.

At the risk of spoiling the punchline for rest of this section, we’ll reveal the big secret: the offline supervised learning setting is often the most relevant in the study of trading and market data. This is because we always have access to a large amount of historical trading and market data, and in many cases we are trying to predict outcomes that are publicly visible in this data (for instance, predicting how much volume will be traded in a certain symbol on a certain date at a certain time). We will delve in detail into the models and fitting methods for offline supervised learning in Section 10, but the important thing to remember here is that you get access to all the labeled training data at once for this technique.

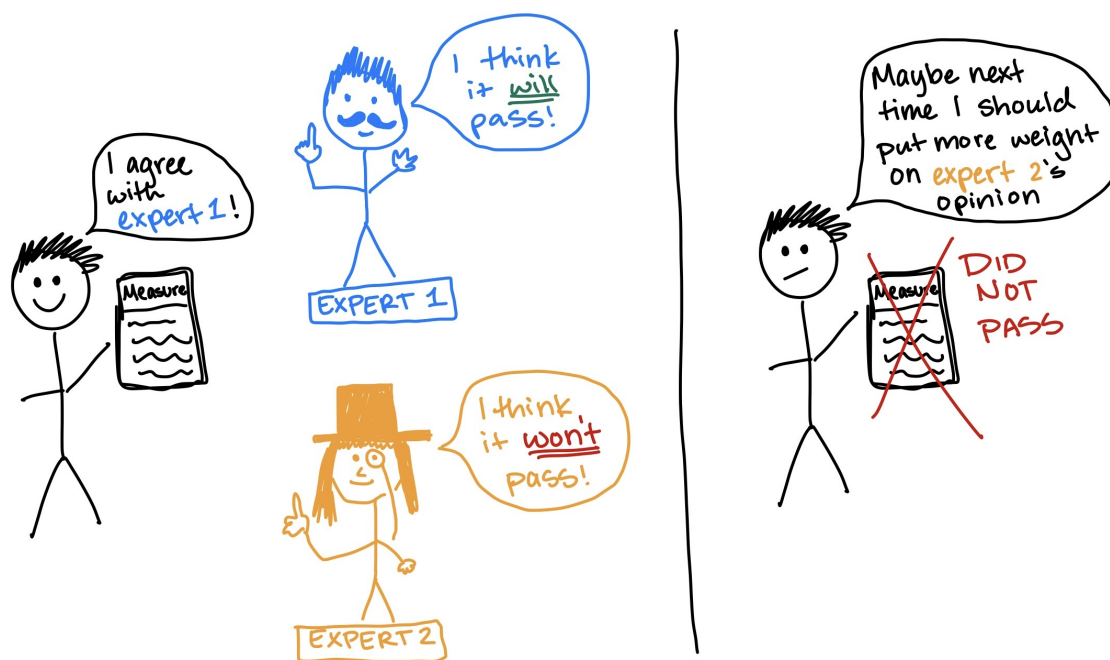
9.1.2 Online supervised learning

In comparison, in *online supervised learning* you similarly have a source of ground truth, but you get access to your training data in a streaming fashion, and may have to make decisions iteratively as you are still collecting data. Online supervised learning can be thought of more as a perpetual spelling test. You sit in a classroom and your teacher reads a word. You try to spell it correctly, and then receive feedback on whether it was correct or not. In this way, you are adapting your learning after every word, rather than studying the whole study set at once beforehand.

Because online learning exists in this streaming fashion, it is easy to (incorrectly) assume that the evaluation of the model happens with each training step, since you are taking a guess at every step before updating the model with the true label. However, even in online

learning, you should be testing your trained model periodically (or once you've deemed the model sufficiently "trained") on data that you did not use to train it. This is because an online learning algorithm could potentially overfit to the specifics of individual data points in the training set if left without separate evaluation. The discussion of how to evaluate specific online learning algorithms is contextual and beyond the scope of this handbook, but it is important to be aware of this when developing online learning practices.

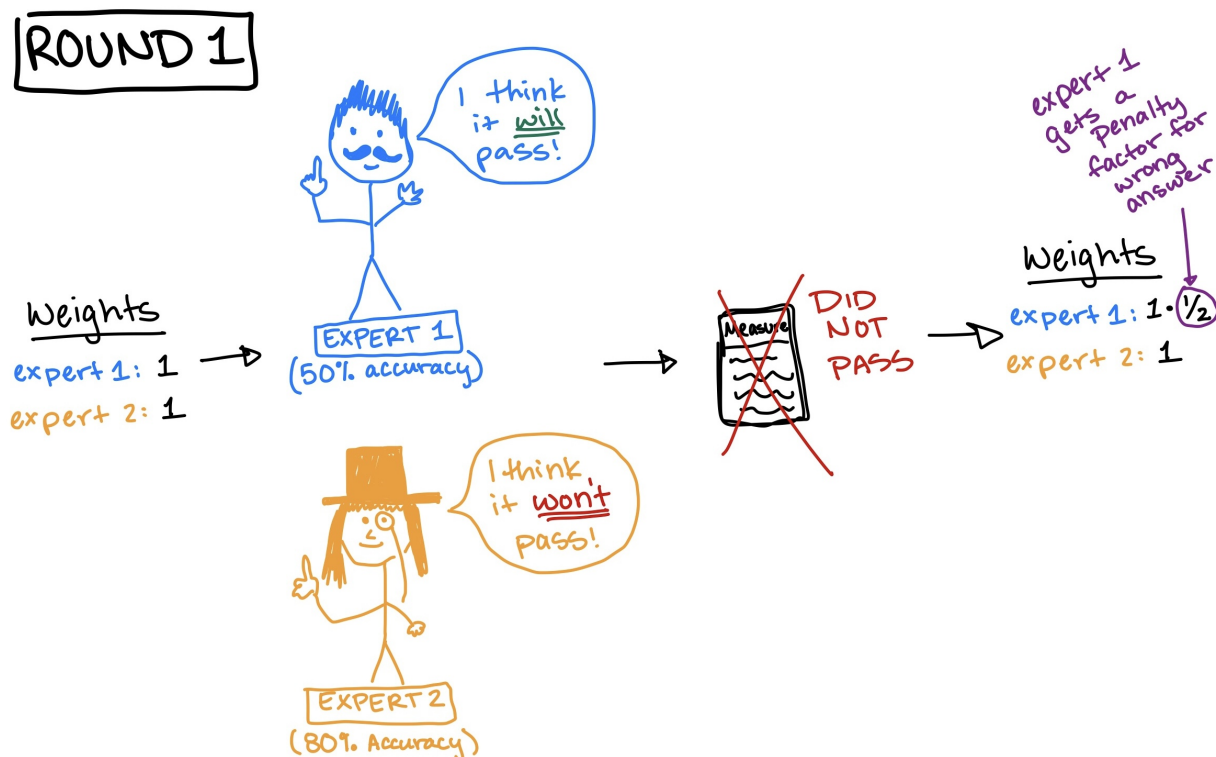
One theoretical framework that is helpful for gaining intuition around this category of techniques is known as the "online learning from experts" problem. In this framing, you are presented with a series of prediction problems. For example, suppose you are a political operative whose job generates a stream of prediction tasks like "predict whether a certain ballot measure will pass." You may have access to advice from several experts, each of whom tells you their own predictions. You can use these to inform your own prediction. After you make the prediction, you learn the true outcome of the ballot measure. Over time, you can improve the quality of your own predictions by listening more to the better experts, even though you don't know a priori who those are.



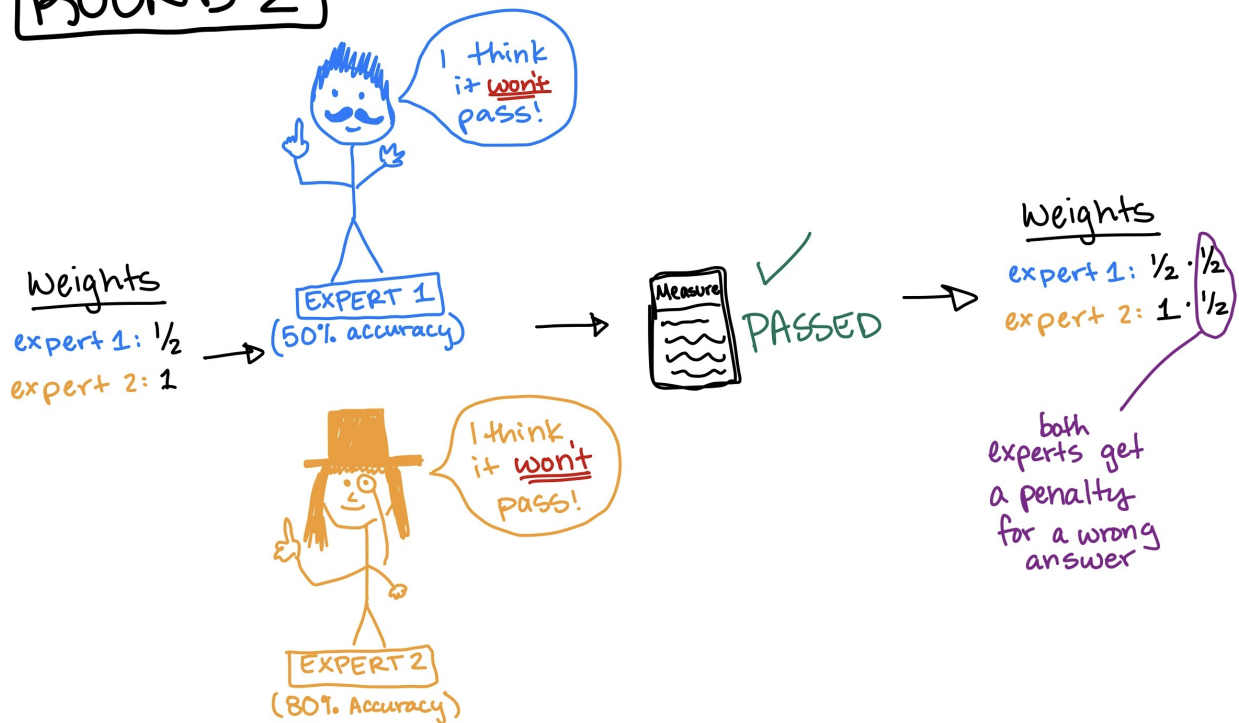
An intuitive way to make predictions is to give a parameter (or weight) to the feedback from each expert. For quantitative decisions, our model can determine its prediction either by taking the weighted average of the experts' answers, or by choosing one expert's answer with probability proportional to its weight. For qualitative decisions, the "averaging" model may not make sense, but the probabilistic selection model still holds. We can initialize our model with equal weighting to indicate a lack of prior information.

After each prediction is made, we learn what the real answer was. This allows us to update the relative weights we place on different expert advice for the future, increasing the relative weights of experts who were right this time and decreasing the weights of experts who were wrong. If we update the weights by multiplying them by a specified factor, this training procedure is known as “multiplicative weights,” a natural approach that has been rediscovered many times. For a survey on multiplicative weights, see [3].

Suppose we have two experts, one with 50% accuracy and one with 80% accuracy and we choose a multiplicative factor of $\frac{1}{2}$. If we start with equal weighting on each expert, after n predictions (treated as independent random variables), what should we expect our weights to look like? For simplicity, let’s not re-normalize them. In this case, we decrease the weight for the weaker “expert” who is no better than random chance will get multiplied by $\frac{1}{2}$ about half the time. This yields an expected value of $2^{-\frac{n}{2}}$. But the better expert’s weight will only get multiplied by $\frac{1}{2}$ about 20% of the time, so the expected weight after n predictions is $2^{-\frac{n}{5}}$. Even if the stronger expert has an unlucky start, it won’t take long for this expected exponential trend to give the stronger expert an irreversible lead. And from that point on, we’ll effectively just listen to the stronger expert.



ROUND 2



ROUND N

expected weights:

expert 1: $\frac{1}{2^{N/2}}$

expert 2: $\frac{1}{2^{N/5}}$

There are a lot of nuances we're skipping over here, including how to define good multipliers for various situations, and how to analyze the convergence of behavior around the expectations. But the goal here is just to give you a little flavor of what online learning entails, and how it differs from other settings of machine learning. That way, when you are faced with a learning problem, you can properly recognize what kind of problem it is, and where you might look for techniques to address it.

In that spirit, let's compare the multiplicative weights algorithm here to a more offline learning-inspired approach. We could simply wait to form any opinion about the experts' relative quality until we have observed a decent sample of predictions. With high likelihood, at this point we would identify the stronger expert and could just listen to her from that point onward. In this case, the potential advantage of multiplicative weights occurs in that

early stretch of predictions, before we have enough sample size to apply the style of statistics we would do on static data sets that are available all at once. With multiplicative weights, we can begin to act on small sample sizes, setting ourselves up to perform better on average than we could with no information, but without committing too soon to conclusions that are based on too few samples to be reliable.

This view is important, because it highlights what online learning does well - it gives us frameworks for incorporating new information on the fly whose information content is substantial but not definitive. This is most useful when we have situations where we are starting from a rather clean slate and will face ongoing consequences of our decisions before we can collect a large data set.

When we say it that way, it sounds like a great fit for financial data problems, especially in trading. But this impression is misleading. For one, we are almost never starting from a clean slate. There is always data from prior days, months, or years that could form an initial understanding.

BAC		MSFT		SPY		
Date	Qty	Date	Qty	Date	Qty	
12/19/25	200	12/19/25	100	12/19/25	0	
12/19/25	100	12/19/25	100	12/19/25	10	
12/19/25	4	12/19/25	5	12/19/25	100	
12/19/25	300	12/19/25	300	12/19/25	100	
12/19/25	100	12/19/25	300	12/19/25	100	
12/19/25	100	12/19/25	10	12/19/25	100	
12/19/25	10	12/19/25	50	12/19/25	10	

BAC		MSFT		SPY		
Date	Qty	Date	Qty	Date	Qty	
12/18/25	100	12/18/25	100	12/18/25	100	
12/18/25	10	12/18/25	100	12/18/25	20	
12/18/25	100	12/18/25	50	12/18/25	100	
12/18/25	30	12/18/25	100	12/18/25	300	
12/18/25	100	12/18/25	10	12/18/25	40	
12/18/25	4	12/18/25	100	12/18/25	100	
12/18/25	10	12/18/25	100	12/18/25	10	

...

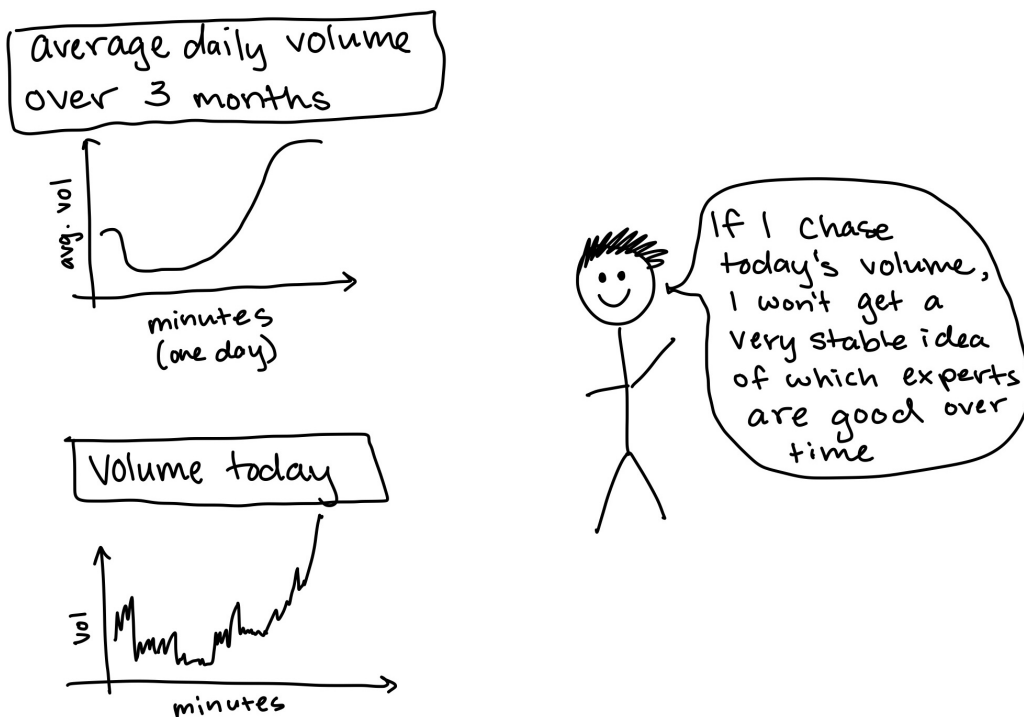
BAC		MSFT		SPY		
Date	Qty	Date	Qty	Date	Qty	
7/7/95	10	7/7/95	50	7/7/95	100	
7/7/95	100	7/7/95	100	7/7/95	100	
7/7/95	4	7/7/95	50	7/7/95	2	
7/7/95	10	7/7/95	100	7/7/95	30	
7/7/95	50	7/7/95	100	7/7/95	100	
7/7/95	4	7/7/95	100	7/7/95	10	
7/7/95	100	7/7/95	300	7/7/95	100	
7/7/95	100	7/7/95	10	7/7/95	100	



Also, the experts/signals/inputs we will be combining to make trading decisions often exhibit much less clarity and differentiation than our example above. For example, we might have two different prediction models (our “experts” in this case), one that offers 52% accuracy and another that offers 54% accuracy.



Most importantly, for weak signals in extremely noisy environments, the actual information value of each incremental piece of data we receive is pretty low, and so weighting it heavily compared to a historical model trained over months of prior data is typically not an advisable thing to do.



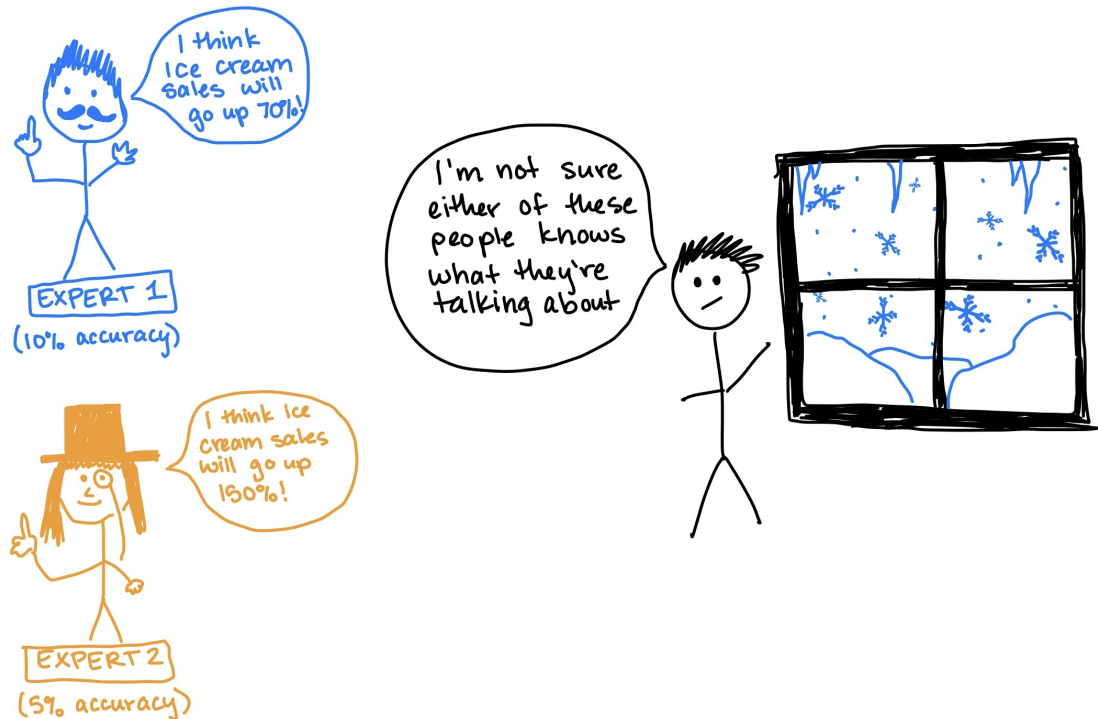
Putting these things together, real time trading decisions don't seem like a great use case

for online learning techniques after all, as the much more significant performance factor is likely to be how well you have learnt from historical data, rather than how well you can shift your model to incorporate the weak signal in real-time noisy data on the fly.

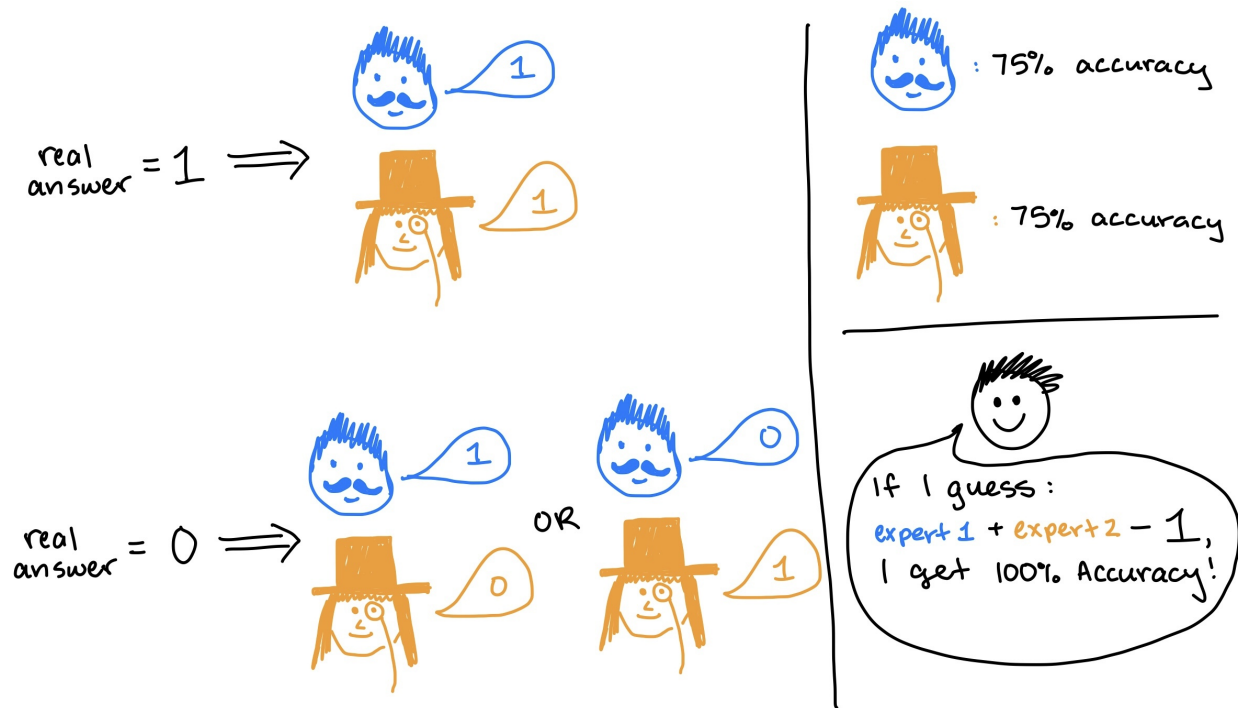
It's crucial to keep in mind here that using offline learning techniques doesn't mean we don't ingest and respond to new data in real-time. It just means that the *shape* and *parameters* of our model aren't really changing in real time. A fixed model can encode things like: "when you see real-time signal X, do Y," it's just not going to adapt the *rules* based on what it's seeing. So balancing online and offline learning techniques well often means figuring out how frequently the rules of the model should really be changing. This answer is heavily influenced by two things: (1) the time scale of evolution of the actual conditions being modeled and (2) the level of noise driving the sample sizes necessary to train effective models. If conditions change dramatically at time scales that are *faster* than we can collect sufficient data to overcome noise, we may not be able to train effective models at all. In financial data, there are good reasons to suspect that many trends are long-lasting enough for us to build effective models for them. But the noise is typically high, and this suggests that periodic retraining of models (say on rolling data sets of days or months of data) may be the right balance, rather than needing online learning techniques that could allow us to shift model shapes more quickly.

It is interesting to compare training algorithms like multiplicative weights to what psychology has to tell us about human decision-making. In many ways, financial data seems perversely engineered to usher us into all of the most insidious mental traps. Firstly, the level of noise makes it kind of like an ink blot, allowing us to see weave whatever narrative we like by focusing on a subset of the data. Secondly, phenomena like the availability and affect heuristics suggest that our decision-making is slanted to overweight recent experiences, and is meaningfully affected by our emotions [1, 18]. Volatile trading environments are perfect playgrounds for cognitive biases to run rampant: you've got human decision makers moving fast, under high stress, responding to constant stimuli. You've also got goals on paper (like maximizing expected profit) that don't precisely correspond to the incentives and disincentives the decision-makers actually experience (like displeasing your boss because you took a justified risk that this time just didn't pan out). An algorithm like multiplicative weights can be tweaked to more closely mimic some elements of human decision-making (e.g. you can adjust the weighting to treat more recent data as more informative than older data), but still imposes some quantitative discipline. One could certainly make a case here that multiplicative weights or similarly dynamic online learning algorithms are psychologically more palatable to human traders and their managers than static models with periodic retraining, as they may represent less of a departure from the way decisions are currently made.

Finally, it's important to note what the learning with experts problem doesn't promise: it doesn't promise that *any* of the experts are any good. The “garbage in, garbage out” maxim certainly applies here - if all of the experts are meh, the results of multiplicative weights will be too. The goal of multiplicative weights is to perform as close as possible to the best expert, as quickly as possible, without knowing a priori who the best expert is.



We might wonder - what about the best *combination* or *function* of experts? Perhaps no single expert is very good, but some function applied to the set of expert opinions would yield better results. To demonstrate why this is possible, let's construct a very artificial example. Suppose we have two experts, and we are trying to predict a binary outcome each time, either 0 or 1. Whenever the real answer is 1, the experts agree and both say 1. But whenever the real answer is 0, the experts disagree, with one randomly saying 0 or 1, and the other always saying the opposite. Let's suppose the real answers are evenly distributed between 0 and 1. In this case, each expert is correct roughly 75% of the time, but we could be right 100% of the time once we notice the simple pattern that expert disagreement means the real answer is 0 and agreement means the real answer is 1. If we suspect something like this may happen, then we may want to define the “experts” that we use an input to an algorithm like multiplicative weights more broadly. We could include simple functions on top of our basic experts, for example, as candidate “experts” themselves.

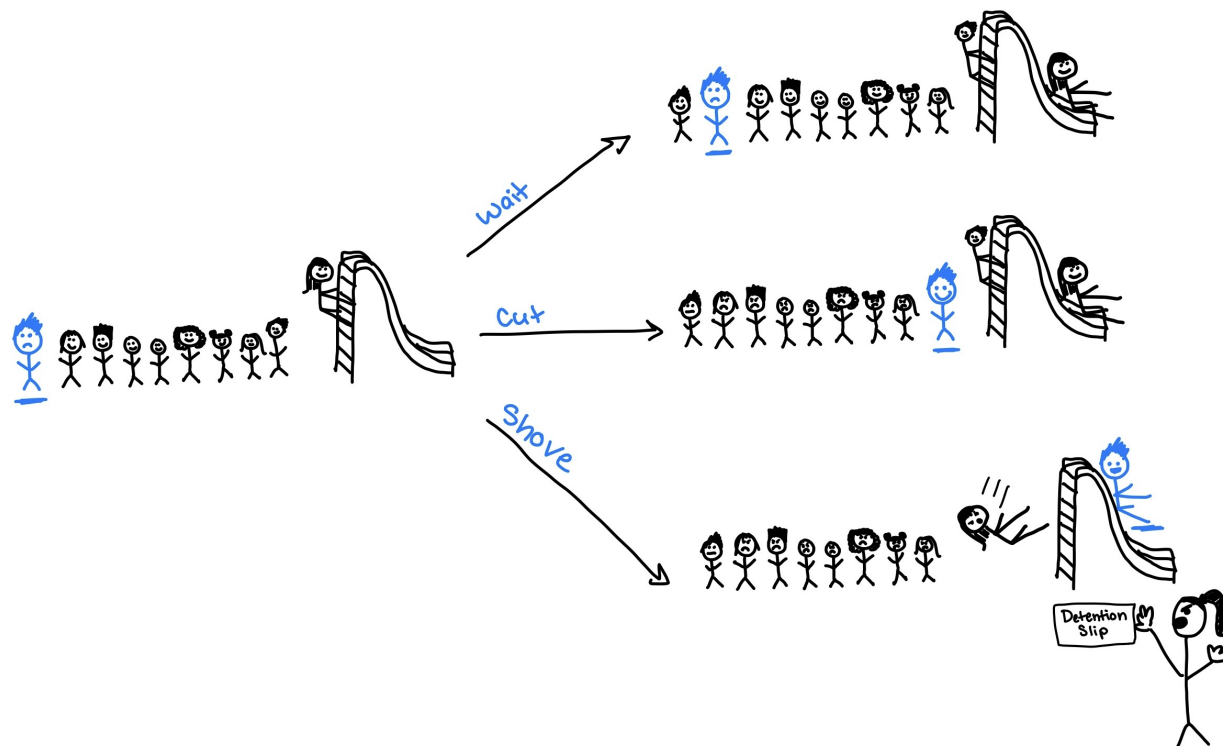


We will not go into more detail here on the vast literature on online supervised learning algorithms, but it is a wide and fascinating field. If you are interested in learning more, one place to start is Blum's survey of the fundamentals of online learning algorithms [5].

9.2 Reinforcement Learning

Let's briefly return to the schoolyard to set the stage for our next topic. In *reinforcement learning*, the modeling and training strategy differs from a spelling test where there is an answer key. We will instead use an analogy of learning social skills and rule-following. Consider a school playground that has one slide and only 5 minutes of recess to ride it. You are tenth in the queue and worried you won't get your turn if you wait. You have a few possible actions you could take that result in various consequences. You could wait your turn and risk the negative consequence of not getting the joy of riding on the slide. You could cut the line to improve your chances of a delightful ride on the slide in this moment but risk the negative consequences of not getting invited to the kid you cut's birthday party (they have a really cool slide at home and now you'll never get to ride it). You could shove the kid currently on the slide off the ladder and enjoy immediate satisfaction of not waiting at all to take your ride, but risk the consequences of getting caught by the teacher and getting punished (perhaps not having access to the slide at recess for a week). The important thing here is that there is no "correct" answer for which action to take in any moment. Instead,

over time and with enough recesses, you will determine what the best course of action is to maximize your long-term joy and minimize the negative consequences. To decide what the best choice is for yourself, you have to try different actions and get a sense of how likely the long-term outcomes are based on your actions.

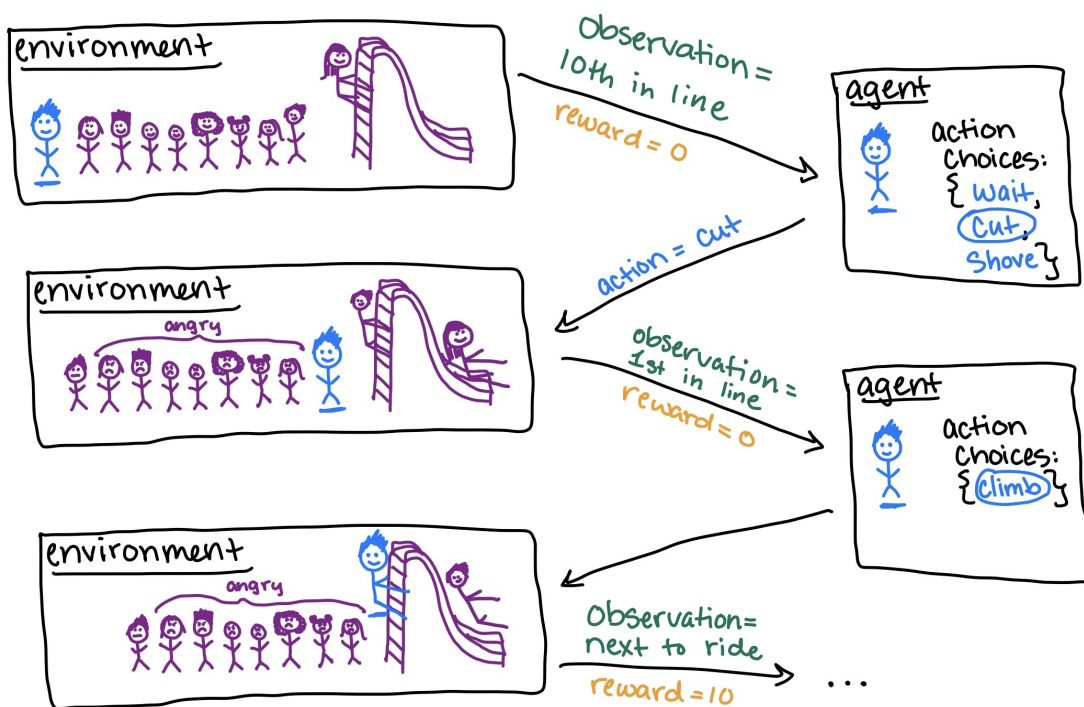


The above scene describes *online reinforcement learning*, which is the typical formulation of the problem. There also exists *offline reinforcement learning* where you have a prerecorded tape of various actions being taken and the subsequent environmental responses, and you are able to “study” which actions lead to which consequences with what probabilities before you are forced to act in the situation yourself. The distinction here is mostly irrelevant for our purposes, and we will assume an online setting going forward, but it is good to know that both forms exist. Reinforcement learning differs from supervised learning in the nature of the feedback the learner (or “agent”) receives. Instead of learning the “right” answer after each decision, the agent gets a reward based on which option they chose. Crucially, they do not learn what the reward would have been for other choices they could have made.

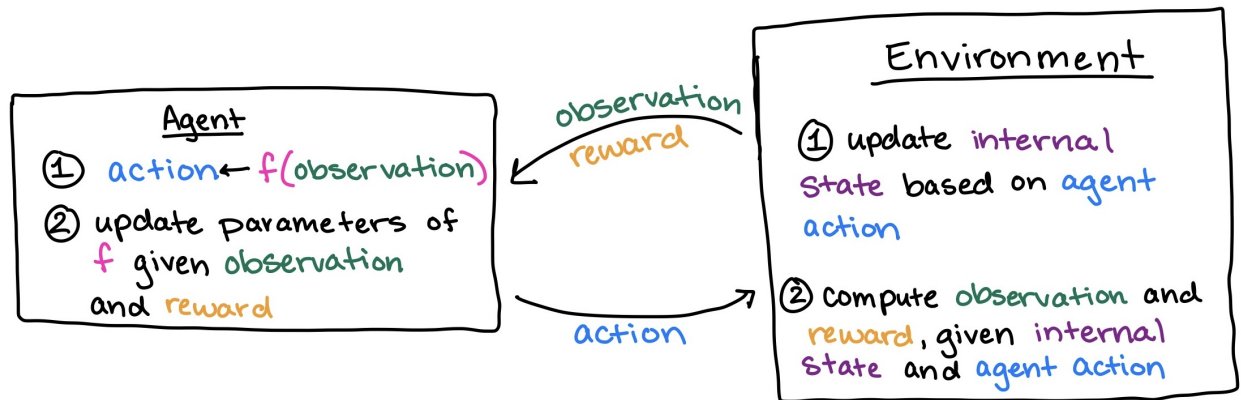
A basic reinforcement learning problem has the same modeling, training, evaluation paradigm as all other machine learning techniques, however this is not typically how reinforcement learning is described. We’ll first go over the typical reinforcement learning formulation and then discuss how the machine learning paradigm applies in this case.

In reinforcement learning, there are two parties: an “agent” and an “environment.” The agent (child waiting for the slide) has a set of available actions (wait, cut, shove). The

agent selects a single action deterministically or with some probability distribution over the available options. The action is sent to the environment, and the state of the environment updates (e.g. if the child chooses to cut the line, the environment would update in that the child would become first in line, but the other kids in the line will feel negatively about the child). The environment then sends a “reward” (or “penalty” if the value is negative) and an “observation” of its internal state to the agent. Importantly, the observation need not be the entire state of the system. In our example of the schoolyard, the observation could simply be the updated place in line of the child, with the social impact on his peers remaining unknown to him. Equally importantly, the reward passed to the agent only reflects the immediate outcome of the choice. In our schoolyard example, the immediate reward of this choice might be entirely positive, since the long-term impact of lowered social standing will not be known until further in the future.



This action $\leftrightarrow$ observation/reward cycle is a hallmark of reinforcement learning, and is normally expressed via this diagram:

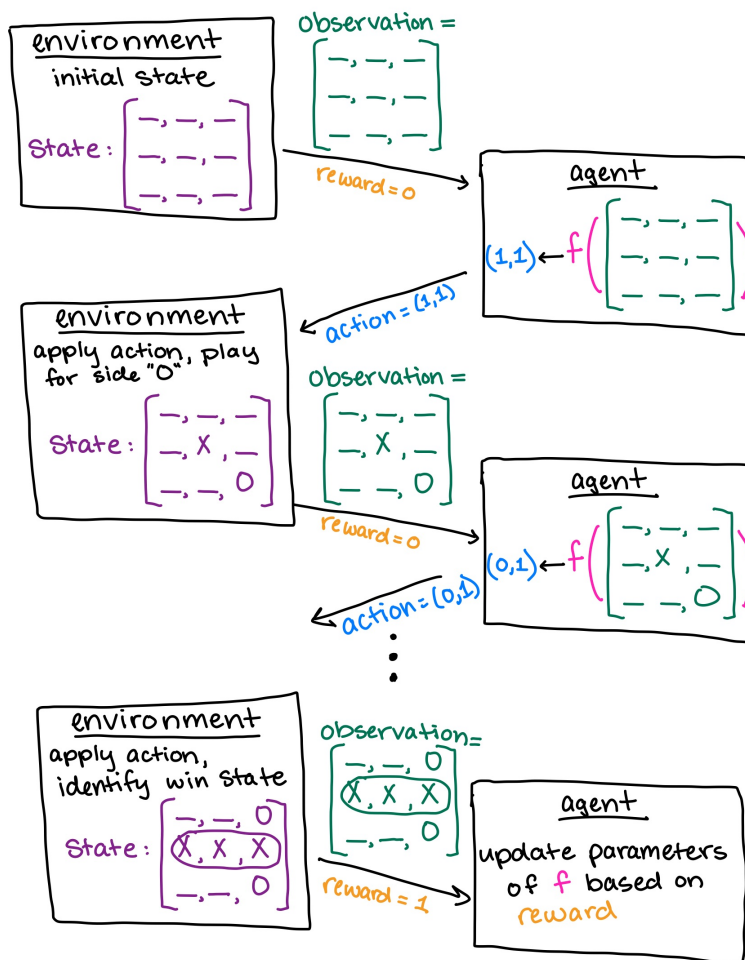


To map this back onto the machine learning modeling/training/evaluation paradigm: the model in this case can be thought of as a function that takes in an observation of the environment at a given time step and outputs an action from a set of possible actions. You can think of this as being the agent itself, or as being a component of the agent (the agent might also hold other components, such as information about prior states of the system). The training procedure is defined as a strategy for updating the parameters of the agent based on the past, current, and potential future rewards. This is normally done through a combination of selecting the best known action based on a given observation, and exploring actions that have not been taken in the past.

Fair evaluation of reinforcement learning algorithms is tricky and context-dependent. When creating a reinforcement learning model, you are not directly drawing information from a set of data or a stream of data. Instead, you are designing a model of rewards, observations, and actions that is based on your understanding of the real system that generates the data you are considering. This means that evaluation will also depend on what you are trying to accomplish. In general, the agent is evaluated based on its average total reward when run on an environment after the training stage has completed. One way that reinforcement learning is evaluated is by training agents on a simplified system model, and then testing it on a more complicated model or interactions with a real system. However, in reinforcement learning it is also common that evaluation is done on the same environment model as training. While this may make sense in some contexts, it can lead to brittle agents that are not robust to changes to the environment or differences between a model and real world behavior.

You may be already identifying a fundamental difficulty of defining and using reinforcement learning: beyond the modeling/training/evaluation paradigm of a typical machine learning problem, we have a meta-problem of modeling the environment, action space, reward space, and observation space. In some cases this is fairly straightforward, however most of the time at least one of these pieces would be non-obvious and require you to make choices that are typically based on your own understanding of the true system.

Let's look at a very simple example: a game of tic-tac-toe. Here, the environment is the current state of the tic-tac-toe game, expressed as a 3x3 matrix of values X , O , or $_$. The matrix is initialized to be all $_$ values. Here, the agent is the player (let's say X). In the first turn, the environment passes an observation of the state of the board to the agent. The agent's action space is all (x,y) tuples to represent indices in the matrix. Let's say the agent starts by playing in the middle space. The environment then updates its state (by placing an O in one of the available slots). The agent is then passed the observation (the updated board), and a reward (in this case, "0" since the game is not over). The agent now has a smaller action space, and can only play in one of the available spaces. The agent chooses an action from the state space and sends it to the environment. This continues until the agent wins (reward= 1), the environment wins (reward= -1), or the game is determined to be a draw (reward= 0). When this happens, the environment sends the agent the final reward and the "episode" terminates.



In this case, the modeling of the framework is fairly straightforward and defined by the rules of the game of tic-tac-toe. However, even in this case, we had to make decisions, like

when the player receives a non-zero reward, and what the values of those rewards should be. Only once this structure is in place can we even begin to talk about how to model and train the decision-making function of the agent.

One of the great drawbacks of reinforcement learning is the amount of design choices that go into setting up the problem. When we think about evaluation in our machine learning paradigm, we think of how to evaluate our trained model (or agent). However, on the meta-scale, we also have to consider how well our choices were made for the environment, actions, rewards, and observations. Therefore, we also have to consider the modeling of our environment, and how to evaluate these choices. Techniques for this type of modeling and evaluation is a further body of research that we won't get into here, but are extremely vital (and often overlooked) parts of determining whether reinforcement learning is appropriate for a given problem.

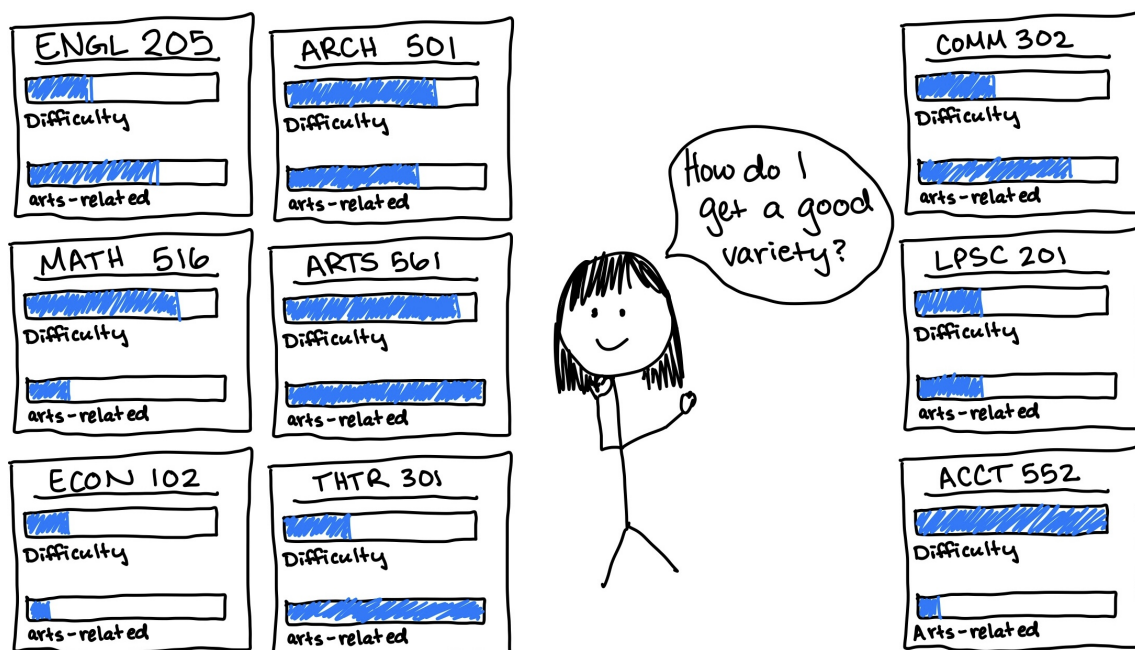
So how does all of this relate to problems involving financial data? It might be a very interesting (and, in fact, very fun) exercise to try to develop a reinforcement learning framework for a market problem. For example, we could try to frame the design of a VWAP (Volume Weighted Average Price) algorithm as a reinforcement learning problem, where the reward function is relative to our benchmark of the market VWAP price, the actions are trade more or less (and how much), and the observations are some amount of prior market data. However, there are two sources of potential failure for this model. The first, as we have discussed, is the minor and major modeling decisions we are making to define these environmental components. The other is that the typical level of variance in outcomes means that it will take a lot of examples for the agent to gain a working understanding of the reward variables, regardless of their training strategy.

Attempting to build this complicated framework to solve the problem may be possible, and may help you to understand the mechanics of the environment you are trying to model. However, in the case of VWAP, it neglects a simpler and likely much less noisy approach where we try to predict and mimic market volume (e.g. following the volume curve) using supervised learning. When considering reinforcement learning as a potential approach to a problem in financial data, it is important to determine how the biases introduced by modeling and the inherent noise in the market might together impact the usefulness of a model trained this way. Though there may be some nice uses for reinforcement learning in financial and market data, in most cases, there is a simpler supervised learning formulation of the problem that handles noise better and is simpler to objectively evaluate.

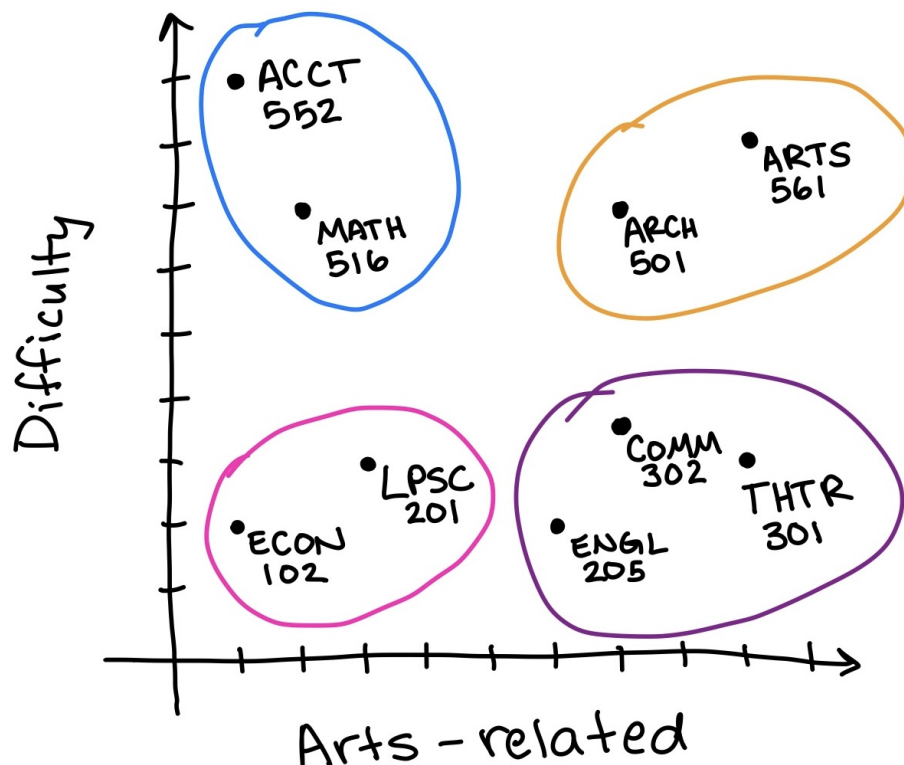
Reinforcement learning is a big topic that we have only briefly summarized here. If you would like to learn more, a good book on the topic is “Reinforcement Learning, An Introduction” by Sutton and Barto [20].

9.3 Unsupervised Learning

Unsupervised learning differs from supervised learning in that there is no “answer key” like in a spelling test, and there is no “reward/consequence” structure like when you are deciding whether to cut in line for the slide. In fact, there is no ground truth at all. Instead, it is more like trying to decide which four classes to choose for an upcoming semester in college. You may want to have a good assortment of certain qualities. For instance, you might want to balance difficulty level of courses, how artistic or engineering focused a course is, and seniority of professors.



One way to do this is by first grouping the course options to identify which courses have similar qualities, and then choosing one option from each category to make sure you are getting a good taste of all there is to offer.



Here, there are no “correct” answers to which classes you should take, as this is up to personal preference on what qualities are important to you to cover, and there is no clear actions or reward structure to create a feedback loop for this decision-making. Instead, you are trying to make sense of a large course catalogue by finding structure in the data.

Unsupervised learning is largely studied in the offline setting, where you have a large data set, and you want to discover its underlying structure. Here, the model, training, and evaluation can look very different depending on our problem. In the example above, our model might be four points and a radius around them that define the four groups that we seek to partition our data into, and our training strategy might be something like k -means clustering (described in Section 9.3.2). Even though there is no objective truth, we still want to evaluate the quality of our trained unsupervised model. However, unlike in supervised learning where we are getting clear feedback about the quality of our model on a set of labeled test points, in the case of unsupervised learning the evaluations are more contextual based on your goals. One example of an evaluation method here would be to find a different set of related qualities to train on and see how many of the courses remain co-categorized.

This degree of freedom can be overwhelming, and there’s not a lot we can say generically here without bringing in the specifics of a given context. But at least one definition of “good” in the unsupervised learning setting is broadly useful, and this involves the notion of clustering, like in our course selection example.

9.3.1 Clustering

Clustering is the fancy machine-learning way of saying categorizing. Naturally, we may want to start organizing our arbitrary mess of data by grouping similar things together, and separating things that are wildly different. The first set of decisions we have to make then, is how to define “similar” and “different.” Our choice of this definition is going to impose some structure on our a priori disordered data universe.

It is most convenient if we can define a metric to measure how close or far two of our data points are from each other. Our choice of this metric will underlie everything else we do, so we should think deeply about our intentions here. For example, let’s say we are looking at prices of trades across different stocks for a particular time period. We may want to cluster stocks into groups based on them having “similar” price movements. If we look at price movements in absolute terms, we will count a 1 dollar price increase in a \$1000 stock as being “similar” to a 1 dollar price increase in a \$1 stock, which doesn’t seem desirable. So we’ll probably want to work with relative price movements. Next, we need to define what time increments we’re going to set for defining our price movements. For concreteness, let’s suppose our data set covers 5 days of trading, and for each day d and each stock s , we’ll compute the relative movement from the opening price $O_{d,s}$ to the closing price $C_{d,s}$:

$$\Delta_{d,s} := \frac{C_{d,s} - O_{d,s}}{O_{d,s}}.$$

This collapses our time series data for each stock into a vector of 5 numbers, where each represents the relative price change from open to close on a single trading day. There are now many ways that we could define a notion of distance between pairs of these vectors. We could take the Euclidean distance, for instance, viewing these vectors as points in 5-dimensional space and computing the length of the straight line that connects them. If we let $d_1, \dots, d_5$ denote our five days and s_1, s_2 denote two stocks, the formula for this distance between the two stocks is:

$$Distance(s_1, s_2) := \sqrt{\sum_{i=1}^5 (\Delta_{d_i, s_1} - \Delta_{d_i, s_2})^2}.$$

There might be other notions of distance that fit our purposes better, depending on what we are trying to do. As a different example, we could define the distance to be a count of how many of the signs are opposite. In this metric, the distance between the vectors $(-0.001, +0.02, +0.0005, -0.01, +0.003)$ and $(-0.0024, +0.0001, -0.003, +0.0012, -0.02)$ is 3.

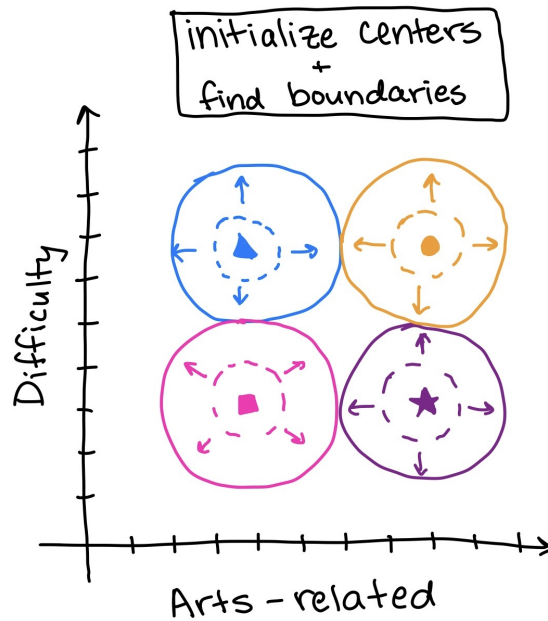
There is a lot of freedom but also there are some rules when it comes to defining a distance (a.k.a. metric). For a function D to qualify as a metric in the formal mathematical sense of metric spaces, it needs to be non-negative, symmetric, declare that the “distance” between any point and itself is 0, and satisfy the triangle inequality. The symmetric criterion means that the distance from a point x to a point y must be the same as the distance from y to x . This is something that seems easy to satisfy, but it is violated by many real-world notions of “distance,” due to asymmetric environmental features like one-way roads. The triangle inequality means that the distance from x to y has to be no greater than the sum of the distance from x to z and the distance from z to y , for any choice of the point z . It is easy to verify that all of these criteria are satisfied for our example that counts the number of differing signs.

It can be challenging to choose a metric, as many seemingly reasonable choices can lead to wildly different results. The Euclidean distance and our sign counting metric, for example, will behave very differently on vectors where all of the values are close to 0, but happen to have opposite signs. There’s not too much we can do about this, other than be mindful of it and make sure our choice of metric best reflects our intentions and that we understand the implications of any edge cases. Once we’ve settled on a metric, we can start organizing our data accordingly.

9.3.2 k-Means Clustering

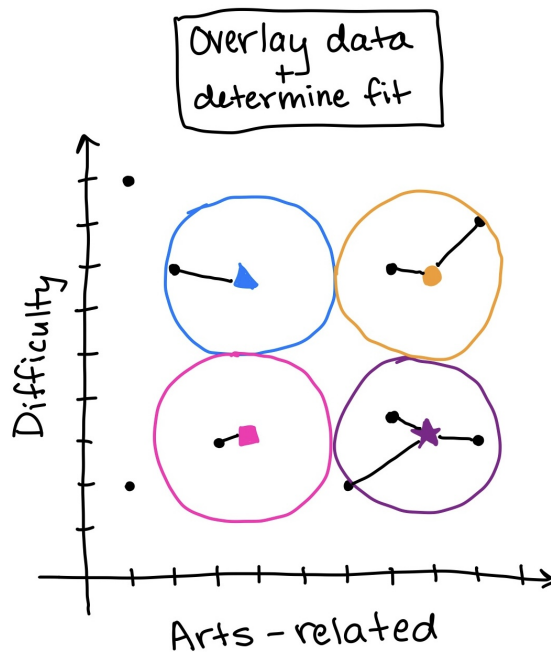
One of the most elegant and common ways to organize data relative to a given metric is k -means clustering. The k refers to how many clusters we want, and the means refers to a step in the algorithm that averages over subsets of our data. The basic idea is fairly intuitive: a good clustering of our data should have the property that data points in the same cluster have typically smaller distances between them than data points in different clusters. In order to achieve such a property, we will start shuffling points around based on their distances from temporary anchor points that describe our current view of where clusters should be. Then we’ll reevaluate our cluster definitions and try again.

More precisely, k -means clustering works like this. We start with k different points in the overall space that we will think of as centers of k clusters. We can visualize this like a map, where we expand the territory around each of these k centers until it runs into another center’s territory:

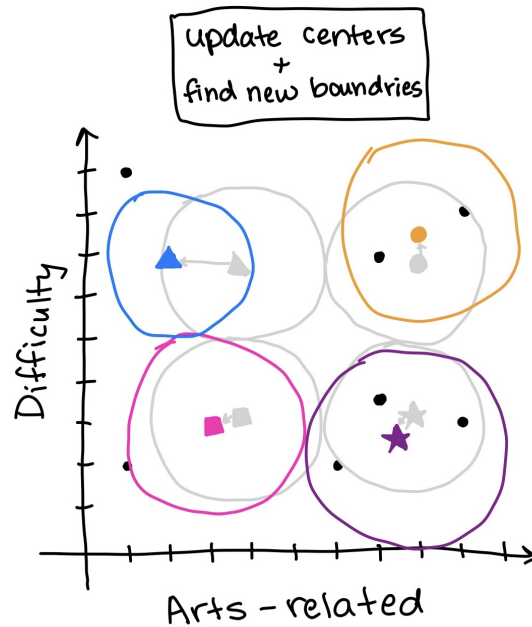


There are many ways to choose these initial k centers. We could choose them randomly from our data set, for example, or smartly, based on some intuition we have about the nature of our data. There is no rule that they have to be actual data points in our set - they could be arbitrary points in the universe of possible data.

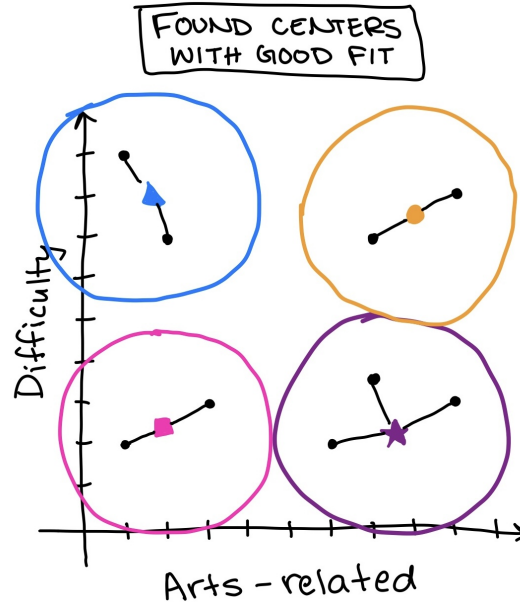
Next, we'll place all of our data points into this map, noting which of the k territories each falls in. In other words, each point is placed in a cluster based on whichever center it is closest to (according to our notion of distance).



Once we've placed all of our points, our so-called "centers" may not look very central anymore. So next, we recompute "centers" for each cluster, either by averaging all of the data points that have been assigned to a cluster (if that makes sense for the way our data is represented numerically), or more generally by choosing a new point that minimizes the average distance to all of the other points in the cluster.



We can now rinse and repeat as many times as we like: once we have new "centers," we can redraw the boundaries and re-place all of the data points. Once we have a new placement, we can compute new cluster centers. The hope is that as we do this over and over, natural clusters start to emerge and become refined. Here is an illustration of what we want to happen:



A few important questions remain. One is: how do we decide when to stop? For this, we can define a notion of quality for a proposed clustering, and track our progress on this from iteration to iteration, stopping when it appears we are no longer making meaningful progress. A natural notion of cluster quality here is to look at the average distances between points in the same cluster vs. the average distances between points in different clusters.

Another question is: how do we choose k ? We might have some idea about what k should be from our context, or we might not. If not, we might try several different values of k , look at the resulting clusters, and make some quantitative or qualitative decision about which value of k best “fits” our data set. Generally, we should prefer lower values of k when possible, as a simpler categorization of our data is desirable if it is equally as good as a more complex one.

It is also interesting to compare the results of k -means clustering to clusters defined by more top-down criterion. For stock symbols, for instance, we could compare the results of k -means clustering on price movements to a categorization that groups stocks according to sectors, or applies a simple formula over inputs like ADV and volatility. If a simple rule for categorizing stocks produces clusters similar in quality to our k -means clusters, we may prefer to use the simple rule rather than the clusters. Even in that case though, we probably can learn some things from the comparison. Symbols that get classified very differently by the two approaches, for example, might be worth digging into to understand why. This is one example of a pretty general tactic for building understanding about data. If you have two ways of doing the same thing, do both! Compare the results. Inconsistencies will often reveal bugs, outliers, artifacts of noise, or other important phenomena.

Unsupervised learning in general and k -means clustering in particular is a well-studied and well-covered topic, so there are many references out there if you want to learn more. One particularly recommended resource for beginners is this video lecture by Broderick [7].

9.4 Machine Learning and Optimization

In some sense, all of the machine learning settings we have discussed in the previous sections are kinds of optimization problems, and machine learning in general is closely related to the mathematical field of optimization. An *optimization problem*, broadly, is a problem that can be broken down into (1) the objective (the expression we want to maximize or minimize over a set of parameters) and (2) the constraints over the parameters of the objective. In supervised learning, for example, we are usually minimizing the “loss”, or distance from our predicted outcomes to the true outcomes over the space of possible parameter settings of our model function. In reinforcement learning, we are optimizing the long-term rewards of an agent acting in an environment, constrained by the types of actions they are able to take.

There are few important things to keep in mind about optimization. One is that is optimization can be very hard as a mathematical discipline: if the objective or the constraints are complicated, it may not be known how to find an optimal solution, or even a particularly good solution. For some simple expressions like lines and for some simple constraints, we can efficiently solve for the best answer. But even for relatively simple objects like trees, we only know heuristic solution methods that seem to find a pretty good answer but do not guarantee that it is optimal. Secondly, the word “optimal” here can be misleading, as it is only relative to the set of objects we specify and the constraints we set. Even precisely solving an optimization problem does not tell us “this a good model.” It only tells us: “of the things we are considering, none of them are better than this.”

We note this connection between optimization and machine learning because, in many cases, we can use well-known optimization methods such as gradient descent to train our models. In some cases, we can also use what we know about optimization to prove the correctness of a training method. This is particularly true in the case of offline supervised machine learning. The field of optimization is expansive, but a good place to start if you want to understand the mathematical underpinnings of the field is the free online Stanford convex optimization course by Boyd [6].

Chapter 10

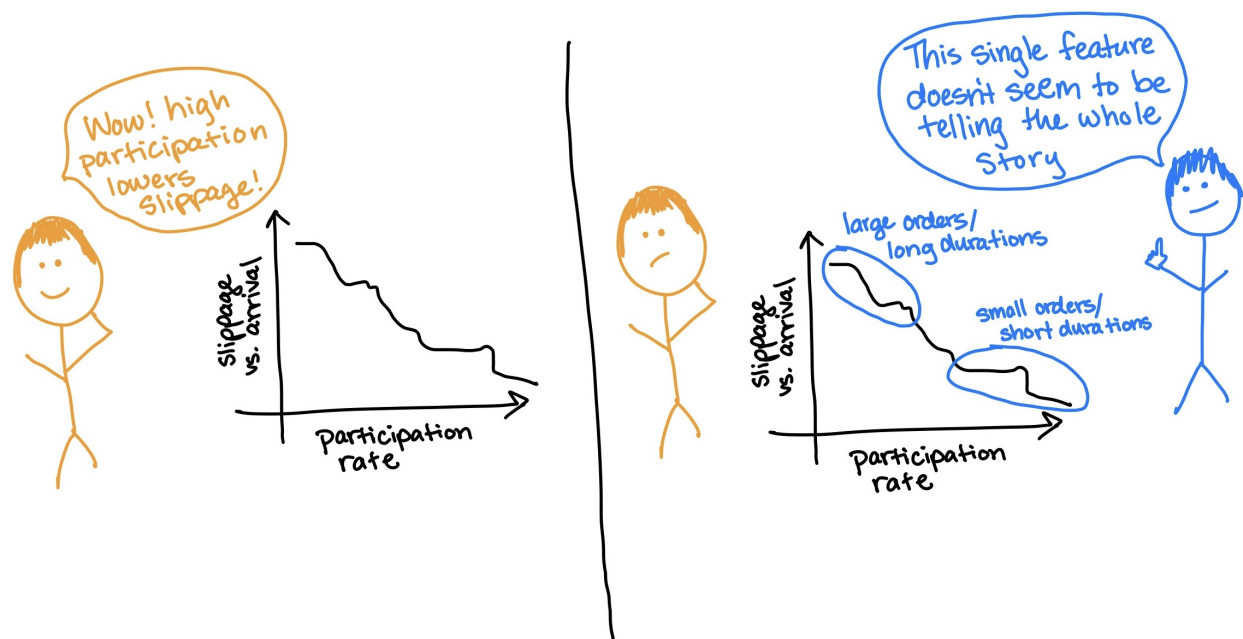
Model Types and Fitting Models

Now that we’ve had a brief overview of machine learning in general, let’s return to the setting of supervised learning, where the goal is to fit a model to a training set of data and we have the “right” answers as a guide. The word model here is just a synonym for function: a map between inputs and outputs. We’re assuming here that we know what kind of outputs we want - we want the right “answers” or “labels,” as they are often called in machine learning. But what’s less clear here is what exactly the inputs to our models should be.

For example, let’s imagine we are trying to build a model that takes in data about an order and spits out a value that we hope is close to its slippage vs. arrival. If this is a pre-trade model, the data we take in should be known before the order starts trading. If this is a post-trade model, used to try to explain how slippage occurred after the fact, then the inputs can include things like *participation rate* that may only be known after the order has completed. Participation rate is defined by looking at the market volume that occurred in that same symbol during the lifetime of the order, and computing what fraction of it was represented by this order. There may be some nuances here, as we may want to filter out trades of various types from the calculation.

There are plenty of opportunities to make good and bad choices about model inputs, which are typically called *features*. Including lots of features can be bad, since it can distract us with coincidences in these features on our training data that do not replicate when we consider fresh data. This phenomenon is called *over-fitting*, and it is especially easy to fall into when data is noisy. Failing to include the *right* features can be bad, as we may miss important patterns or misinterpret partial patterns that we do find. For example, suppose that we include participation rate of an order in our model of slippage, but we do not include features of order size or duration. We might then find that higher participation rates are associated with less slippage (e.g. more favorable trading outcomes). But we should not conclude from this that trading faster leads to better outcomes! This can be a classic

example of correlation not equaling causation. Participation rates are often high for small orders, as they can complete quickly without a lot of other trading happening in the same time frame. The real causation may be: smaller orders have less price impact and hence less slippage, and they also have high participation rates. But if you fix a given order size and increase the participation rate, we shouldn't expect this to *reduce* slippage.



Another pitfall to keep in mind is the inclusion of features that are too good. More specifically, features that are co-dependent with the outcome may lead to models that are misleading or at least not very useful. As a silly example, we could include slippage as a feature in a post-trade model of slippage. We'd probably get a great model! But we'd learn nothing. Things this extreme feel easy to avoid, but there is often a wide universe of potential features that are intertwined with our outcomes in non-obvious ways.

At its best, feature selection is a mix of science and art: human intuition and exploratory data analysis often goes into crafting features that are a succinct encapsulation of what we think is most relevant about our data, leading to the construction of useful and interpretable models. It's hard to say much more generically about feature selection, as it's highly dependent on the data and the context. But it is perhaps the most crucial step of model building, and also the least systemized. For now though, let's discuss the further steps of model construction, assuming we have selected a reasonable set of features of our data to serve as inputs.

To pick a function that best “fits” the relationship between our features and the labels for our data, we need to establish several things:

- a set of candidate functions
- a way of grading each candidate function on how well it fits the data
- a way of searching through the candidate functions to try to find better ones

In the simplest case, you may have a set of candidate functions where you can afford to try *all* of them. This can happen particularly if the set of features and answers you are mapping between is very small. For example, suppose our data points are just strings of 3 binary values, and our “answers” are just single binary values. In this case, there are $2^3 = 8$ possible data values, and 2 possible answers. An arbitrary function between data values and answers can then be defined by specifying 8 outputs for the 8 inputs. There are exactly 2^8 different ways to do this, since we have two choices for each of 8 things. This is a number of possible functions that we can search exhaustively. Most of the time though, considering all possible functions and/or exhaustive search through our chosen set of functions is not practical. In the following subsections, we will summarize some of the most common choices for candidate functions, how they are graded, and how they are searched.

We’re using the term “graded” here in a way that may be confusing: it’s tempting to think that how well a function fits our training data represents a full picture of how good it is as a model for capturing real phenomena. But this requires a notion of *generalization*, meaning that our model generalizes to perform well on data that it never saw during training. Generalization is what separates a model that somehow memorizes answers to specific examples from one that is picking up on more fundamental patterns. Testing how a model performs on fresh data is a topic we will tackle in the next section, but for now, we’ll just talking about grading a model in terms of its success on our training data in particular, as a way of choosing a model among a large set of options.

We’ll use volume curve prediction as a running example. The term *volume curve* refers to the function that maps a time t during the trading day to the percentage of volume for that day that had traded as of time t . If we limit ourselves to volume that trades during normal market hours, from the open around 9:30 am through the close around 4 pm, this function will be 0 at 9:30 am and will be 1 after the closing auction. We’ll denote this function by VC . Thus, $VC(t)$ represents the volume curve at time t . We compute this on a per symbol per day basis, so we will have to specify what symbol and day we are working with in order to fully define this function.

The exact value of VC at some intraday time t cannot be known until after the close, as we can’t yet know the total volume for the day. But we can try to make predictions for $VC(t)$ using things we know as of time t . For now, we’ll stick with two basic inputs. One will be the historical average for $VC(t)$ for this symbol over the preceding 20 days, evaluated at

the same time t . We'll denote this by $HVC(t)$. (The H stands for "historical.") The other will be the volume traded so far, divided by the average daily volume for this symbol (also computed over the preceding 20 days). We'll denote the volume traded so far as $V(t)$, and the average daily volume as ADV . Here is a table of these attributes and their meanings for reference as you move through the model types:

$V(t)$	Cumulative volume at time t for a given day.
ADV	Average daily volume (over past 20 days).
$VC(t)$	Volume curve at time t . Computed: $\frac{V(t)}{\text{Total Day Volume}}$ This is our prediction goal.
$HVC(t)$	Historical volume curve at time t . Average $VC(t)$ over the past 20 days.

Generally throughout this chapter, our model inputs are $HVC(t)$ and $V(t)/ADV$. Our goal will be to find a function of $HVC(t)$ and $V(t)/ADV$ that best fits our desired value of $VC(t)$ over a training set of historical data.

10.1 Lookup Tables

A simple (and often effective!) approach is to make a lookup table that lists an output value for each possible set of input values (a.k.a features). This corresponds to considering *all* possible functions over the inputs, as there is nothing that constrains the output values or forces them to have any common structure. As mentioned above, this tends to work best when the domain for the function is small. In our running example for volume curves, we might make a lookup table with three columns: one for possible values of $HVC(t)$, one for possible values of $V(t)/ADV$, and one for our prediction for $VC(t)$. The size of such a table will depend heavily upon our level of precision for the feature values $HVC(t)$ and $V(t)/ADV$. If we round each feature to a relatively coarse set of values, our table will be smaller in size (representing a smaller set of possible functions), and we will have a lot of historical examples for many of the combinations of rounded values. If we round each feature to a relatively fine set of values, we will have a larger table and corresponding set of possible functions, and much smaller sample size per item in the table.

Once we've chosen a rounding or other discretization of our features, our task becomes: how to set the predicted value of $VC(t)$ for each combination of (rounded or otherwise discretized) values of $HVC(t)$ and $V(t)/ADV$ in our table. A natural approach is to take a weighted average of the real values of $VC(t)$ that occur in our training data for sym-

bols/days/values of t such that $HVC(t)$ and $V(t)/ADV$ take on these values. For example, let's say that we round $HVC(t)$ to the nearest multiple of 0.1 and $V(t)/ADV$ to the nearest multiple of 0.1. So we'll have rows in our table for cases like $HVC(t) \approx 0.3$ and $V(t)/ADV \approx 0.4$, and we'll need to decide how to fill them in.

To make a prediction for $VC(t)$ at moments where $HVC(t)$ rounds to 0.3 and $V(t)/ADV$ rounds to 0.3 for example, we can take the weighted average of $VC(t)$ in such moments over a training set of historical market data. We might choose to weight evenly for data points from the same symbol and weight across symbols according to total notional value, for example - there are several reasonable choices here. We'll keep things somewhat simple by weighting each data point by the notional value of trading it represents. In other words, we'll add up the number of dollars traded in each symbol/time period, and weight the corresponding data point proportionally. This approach treats time periods with heavier trading (in terms of notional value) as more important.

We can then fill in our table with our empirical results.

Just for illustrative purposes, let's do this for a small data set of trading volume in the symbol BAC over the time period from January 1, 2022 through January 31st, 2022. Here's a partial view of the table we obtain:

Out[18]:

	V/ADV	HVC	predicted VC
0	0	1	0.045870
1	1	1	0.086511
2	1	2	0.151322
3	2	1	0.146904
4	2	2	0.177977
...	...	...	...
61	13	8	0.856216
62	13	9	0.898541
63	14	9	0.931976
64	14	10	0.987631
65	15	10	0.986898

Here, we have rescaled our rounded values to display as integers. So a "1" in our table for HVC means $HVC(t)$ rounded to 0.1.

Let's quantify how well our table performs as a prediction model for $VC(t)$ as a function of $HVC(t)$ and $V(t)/ADV$. For this, we'll want to test on *fresh* data that wasn't part of

our process for making the table. Since we used data from January 2022 to make the table, let's test on data from February 2022.

For our test data, we'll collect data from BAC in the same fashion, recording a data point once every 10 minutes of the trading day. For each test data point, we'll round the observed values of $HVC(t)$ and $V(t)/ADV$ and see if this combination appears in our table. If so, we'll use the predicted value in the table as our prediction for $VC(t)$. If not, we'll use the observed value of $HVC(t)$ as our default prediction.

This is what our test data now looks like:

Out[28]:

	V/ADV	HVC	SYMBOL_NAME	HVC_raw	real VC	NV	predicted VC
0	0	1	NYSE_TQ::BAC	0.059776	0.055521	8.906944e+07	0.04587
1	0	1	NYSE_TQ::BAC	0.060165	0.035452	9.533267e+07	0.04587
2	0	1	NYSE_TQ::BAC	0.057622	0.040444	6.563492e+07	0.04587
3	0	1	NYSE_TQ::BAC	0.056153	0.047044	7.623148e+07	0.04587
4	0	1	NYSE_TQ::BAC	0.056037	0.052289	9.720714e+07	0.04587

The “HVC raw” column here preserves the higher precision value of HVC before we rounded it and convert it to an integer to make the HVC column match with our trained table. This is used as our baseline prediction, and is also copied into our “predicted VC” column whenever our lookup table fails to have a matching entry.

Now we will compute the mean squared error for our predictions using the table, and we'll compare this to the mean squared error of baseline historical predictions (HVC raw). Using our table, we get a mean squared error that is about 1.13 times as big as the baseline.

So... what's going wrong? Why are our predictions *worse* than the baseline historical values? There are several possibilities. We could be under-training (not using enough data to form our table). We could be choosing our level of rounding poorly. We could be investigating a feature (V/ADV) that is not particularly helpful, etc. Before giving up on this feature though, let's adjust some of our other choices and see if we can improve our results.

First, let's just increase the amount of training and testing data. This time, we'll train the table on BAC data from all of 2021, and we'll test on BAC data for all of Q1 2022. This time the mean squared error is ... about 1.18 times the baseline.

Since throwing more data at the problem made things worse instead of better, we may suspect that this is not a simple issue of under-training/overfitting. Let's see what happens now if we keep this greater amount of data, and we change the rounding of HVC we use

in our table. Instead of rounding to the nearest multiple of 0.1, we'll round to the nearest multiple of 0.01. This is going to mean that we make more distinct predictions (each row in our trained table now corresponds to a narrower range of values), but also we will tend to have fewer data points as a training sample corresponding to each row. Nonetheless, doing this gives us a better, though still worse than baseline result. Now we get a mean squared error that is about 1.09 times the baseline. If we up the amount of data again, training on 18 months leading up to Jan 1, 2022, and testing on Q1 and Q2 in 2022, our performance degrades drastically to 1.59 times the baseline.

This demonstrates how frustrating it can be to try to build a model on a per symbol basis. Increasing our data collection window, which should be better for model performance in static conditions, also means training/testing over increasingly disparate market conditions and external factors that shift over time. It's also pretty limited in scope - getting 10 times as much data in this way, for example, is impractical in most situations. This is a core reason why normalizing over symbols and building up larger data that is *contemporaneous* in time can be a valuable approach. Let's try that here - we'll train on data from Q4 2021 only, but now we'll do it for the top 2000 symbols by notional value, rather than just BAC.

This will require us to make some decisions for how we average our training data over symbols. We'll again keep things relatively simple and weight each test data point proportionally to the total notional value traded in that symbol over that 10-minute time period. We'll test over Q1 2022, using the same set of 2000 symbols.

Now our mean squared error is only about 0.80 times the baseline! We might wonder - perhaps the performance of our model is still poor for BAC in particular, but performs well overall. Actually this is not the case - even testing only on BAC now, we obtain a mean-squared error that is about 0.92 times the baseline.

This demonstrates the power of normalization as a tool for amplifying training data size by casting a wide net across symbols - the quality of a model we train over many symbols is often far superior to one trained only on a single symbol.

Next we'll investigate what happens as we fix these same features, but we change the form of our model. Putting pure empirical results in a table model like this is appealing because the numbers are inarguably real, and they don't come with pre-conceived notions baked in. However, they can be erratic - pulled around by outliers and fleeting phenomena, and they might violate common sense properties that we would expect to see. An empirically derived table in this style functions somewhat like a child that doesn't yet draw connections between things across settings. They might know from experience that they don't like pizza or quiche, but they have not figured out yet the underlying reason that they don't like cheese. This approach is entirely stumped when it encounters a specific situation it hasn't seen before.



This is the flip side of not assuming some underlying structure that connects the training examples - we have freedom from bias, but we are held back by treating every distinct situation as a new problem to be solved from scratch. This is also a large part of what is driving our need for tons of data here.

10.2 Linear and Polynomial Models

Let's consider an opposite extreme: fitting a model from a very limited function class. The classic example here is linear regression. When we perform linear regression over a set of input variables, we are simply defining the class of functions here to be the set of linear functions. In our running example, this means we are considering functions of the form:

$$c_0 + c_1 HVC(t) + c_2 \frac{V(t)}{ADV}$$

where c_0 , c_1 , and c_2 are constants. Our goal is to choose the values for the coefficients c_0, c_1, c_2 that make this quantity fit best with the true values of $VC(t)$.

If we think of c_0, c_1, c_2 as arbitrary real numbers rather than having bounded precision, then our class of functions here is still infinite. But it is clearly much more restricted than the class of *all* functions. Instead of having freedom to fill in each entry of a large table

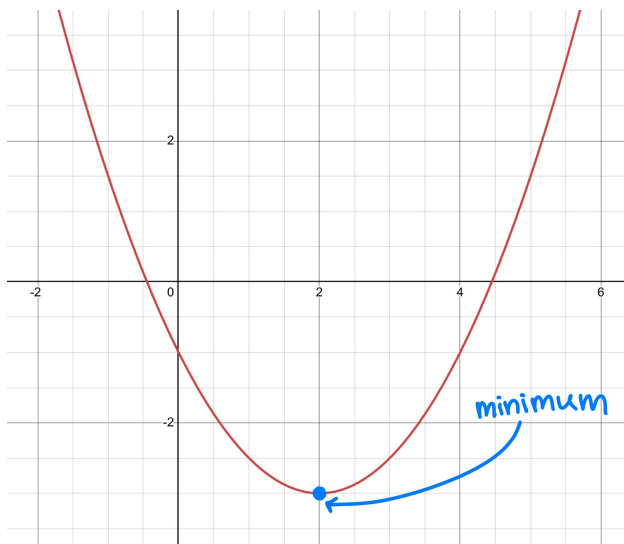
however we want, we are now only free to set three coefficients.

So how should we set them? We can start by defining an error metric for how relatively good/bad a particular choice of coefficients is. A common choice here is again the mean-squared error metric, which computes

$$\left(c_0 + c_1 HVC(t) + c_2 \frac{V(t)}{ADV} - VC(t) \right)^2$$

for each data point, and then averages these values over data points (with some reasonable choice of weighting). One consequence of the squaring in this metric is that cases where the candidate model underestimates $VC(t)$ contribute the same as cases where the candidate model overestimates $VC(t)$. In this example, this feels like a desirable feature. In some applications of linear regression though, we may want to change this, as underestimating might be in some sense worse for us than overestimating (or vice versa).

If we fill in the values of $HVC(t)$, $V(t)/ADV$, and $VC(t)$ from our training data, then the mean squared error becomes a quadratic function of the variables c_0 , c_1 , and c_2 . Note that the fact that there are three variables here is particular to our example, but the fact that the mean squared error of a linear function is a quadratic function of the coefficient variables is true more generally. It's not very hard to find the minimum of a quadratic function. With one variable, this is easy to see:



Mathematically, that minimum point corresponds to the place where the derivative of the function is 0. Since taking derivatives of quadratic functions is easy and yields linear functions, we can generally solve for the coefficients that minimize the mean squared error over our training data. Solving this directly as a system of linear equations avoids us having to search through potential values of c_0, c_1, c_2 to see what works best.

This is called *ordinary least squares regression*, and it is implemented in scikit learn, for example. Here's what it looks like when we fit a basic linear model to our volume data for BAC for Q4 2021 (without the rounding that we employed in the lookup table model above):

```
In [32]: 1 X = data[['V/ADV', 'HVC']]
          2 y = data['predicted VC']
          3
          4 linear_regressor = LinearRegression()
          5 linear_regressor.fit(X,y,sample_weight = data['NV'])
          6
          7 #Let's look at the coefficients and y-intercept of our linear function.
          8 #the coefficient for V/ADV will appear first, then HVC, then the y-intercept
          9 linear_regressor.coef_, linear_regressor.intercept_

Out[32]: (array([0.06004939, 0.90574016]), 0.01181800734461691)
```

Thus, our linear model is roughly:

$$0.012 + 0.060 \frac{V(t)}{ADV} + 0.906 * HVC(t) \approx VC(t).$$

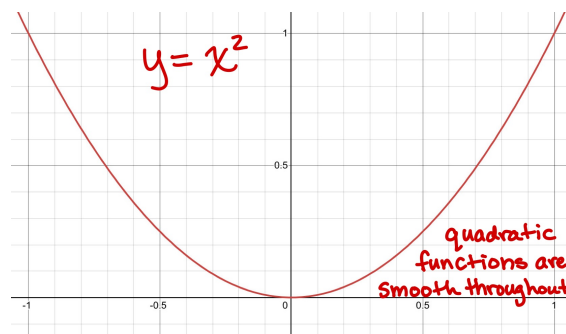
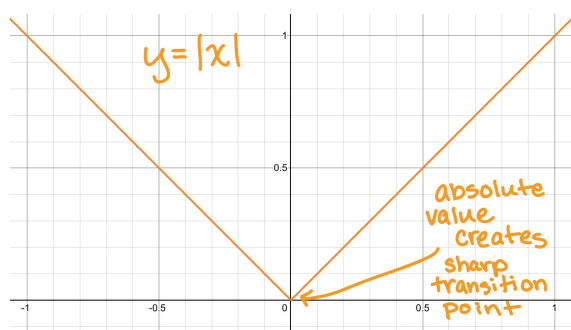
Let's see how this performs for test data for BAC over Q1 2022. In this case, we get a model that outperforms the baseline already, with a mean squared error that this is about 0.94 times the baseline. This is in line with what we should expect: a linear model can still perform reasonably with a small training set since the number of parameters we are fitting is small. However, the overall performance here is worse than the table model trained over 2000 symbols.

Let's see what happens here when we train our linear model over those same 2000 symbols. In this case, we get a linear model of:

$$0.02148 + 0.00007 \frac{V(t)}{ADV} + 0.96509 * HVC(t) \approx VC(t).$$

This model is getting very close to just copying $HVC(t)$, so we should not expect it to provide much improvement over the baseline.

You might wonder why it's common to square the error quantity $c_0 + c_1 HVC(t) + c_2 \frac{V(t)}{ADV} - VC(t)$ instead of taking its absolute value. Afterall, taking the absolute value would also have the feature that underestimates and overestimates are treated equally. However, absolute values introduce sharp transitions when the sign changes from positive to negative or vice versa, and such transitions cause trouble for defining derivatives. This can be annoying to deal with while solving for optimal fits. In this way, squaring is cleaner and leads to smoothly differentiable functions.



Linear functions are a relatively simple and often useful set of candidate functions. They are good at expressing monotonic and consistent relationships between variables. If our data does ultimately exhibit a roughly linear relationship between the variables we've chosen, narrowing our focus to fitting these coefficients will greatly help us block out noise and find what we're looking for. But - there's no guarantee that such a linear relationship exists. In our volume curve example, it seems like linear relationships for this limited variable set do not capture anything better here than the baseline relationship between the historical value $HVC(t)$ and $VC(t)$. Since we found better results with a lookup table, we know that we can continue looking for non-linear relationships in these variables to improve our prediction quality.

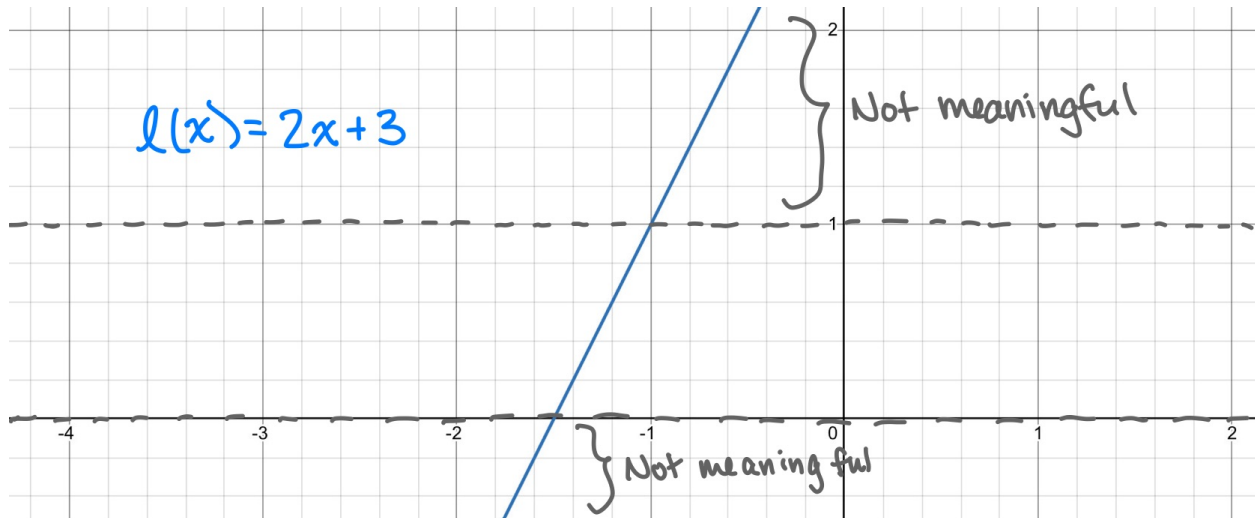


10.3 Logistic Models

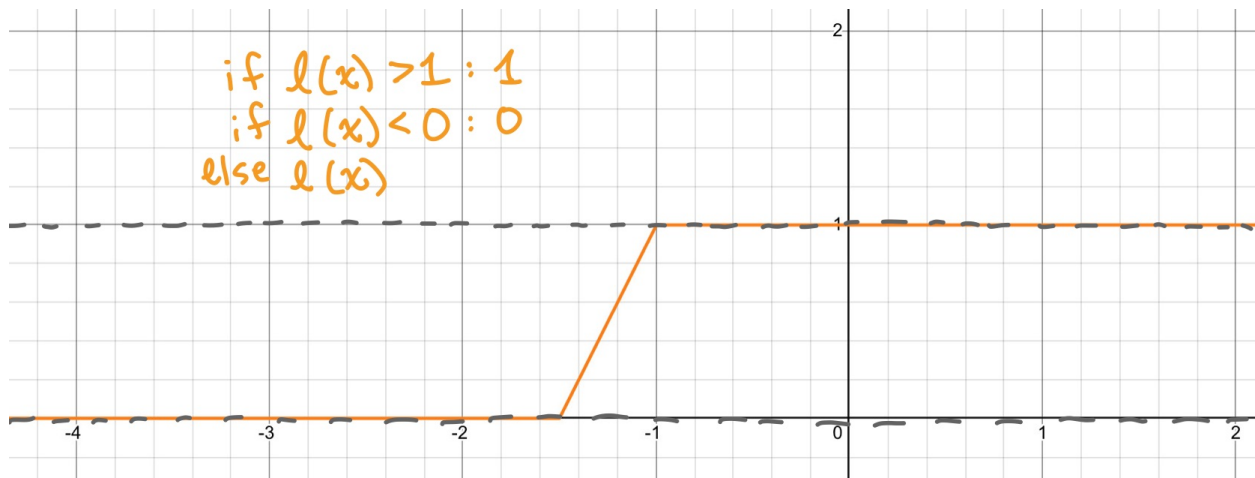
Sometimes we want to model a relationship that involves a strictly bounded quantity, like a percentage or probability, and an unbounded quantity, like trading volume. In fact, the quantity we've been considering here, $VC(t)$, is bounded between 0 and 1, while one of our

features, $V(t)/ADV$, can theoretically be arbitrarily big! This means our linear model above will start predicting nonsensical values for $VC(t)$ when $V(t)/ADV$ gets large enough.

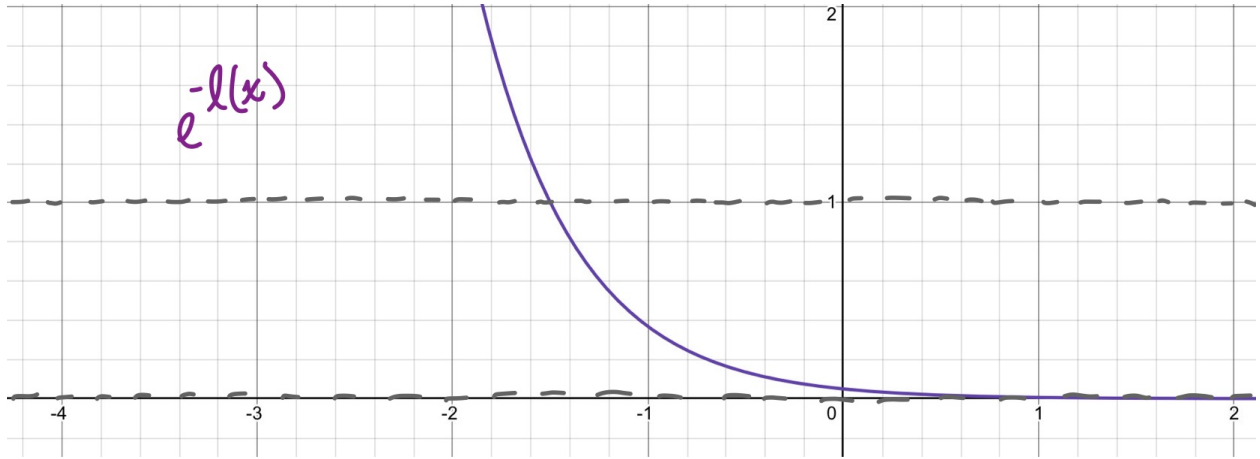
There are many possible ways to deal with this behavior. Let's use an arbitrary example line $\ell(x)$ to illustrate the options:



We could, for instance, cap our model's output to sensible values by replacing every output > 1 with 1 and every output < 0 with 0:



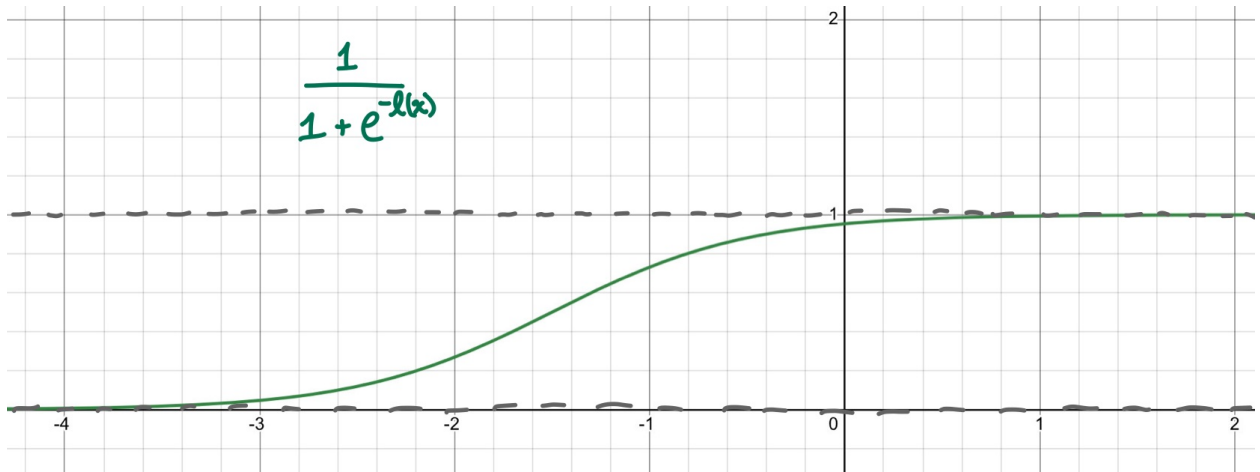
This achieves our high-level goal, but there are potentially smoother adjustments we could make. Let's start from first principles: how might we take a line that can go from $-\infty$ to ∞ , and reshape it to stay inside the range 0 to 1? We can do this in several steps. First we'll take our line $\ell(x)$ and put it in the exponent, considering instead $e^{-\ell(x)}$:



When $\ell(x)$ is 0, $e^{-\ell(x)}$ is 1. When $\ell(x)$ is very negative, $e^{-\ell(x)}$ is very positive. When $\ell(x)$ is very positive, $e^{-\ell(x)}$ is near to 0. Thus, we have reshaped things by mapping 0 to 1, and mapping positive infinity to 0. This makes our value $e^{-\ell(x)}$ stay in the range of 0 to 1 whenever $\ell(x)$ is positive. We may want a function though that increases from 0 to 1, as $\ell(x)$ goes from $-\infty$ to $+\infty$, rather than decreasing from 1 to 0 as $\ell(x)$ goes from 0 to $+\infty$. For this, we can define:

$$p(x) := \frac{1}{1 + e^{-\ell(x)}}.$$

Now, when $\ell(x)$ is very negative, this denominator is very large and positive, and hence $p(x)$ is very small. In other words, as $\ell(x)$ approaches $-\infty$, $p(x)$ approaches 0. As $\ell(x)$ approaches positive infinity, $p(x)$ approaches 1:



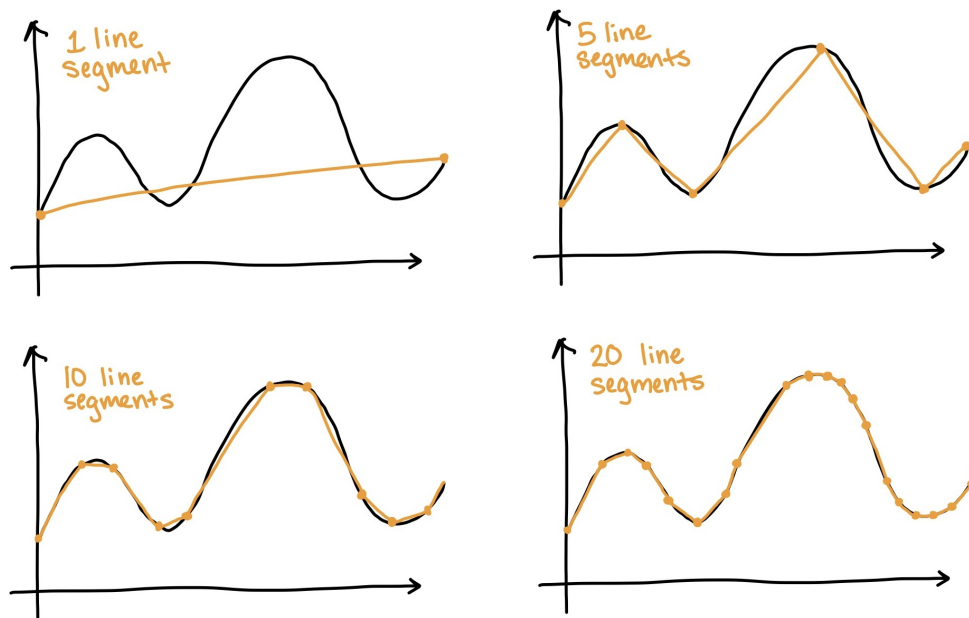
Using this shape for $p(x)$ is typically referred to as logistic regression. This means we would rework our model to find the coefficients c_0 , c_1 , c_2 that give us the best approximation of the form:

$$\frac{1}{1 + e^{-c_0 - c_1 HVC(t) + c_2 \frac{V(t)}{ADV}}} \approx VC(t).$$

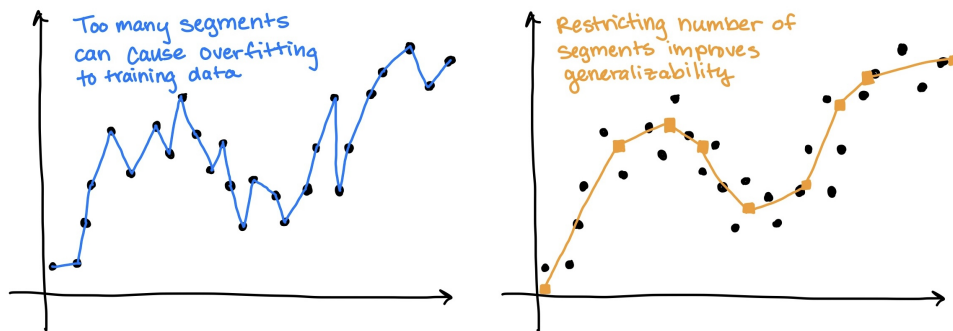
Note that doing this with an out-of-the-box logistic regression tool may require some hacks, as logistic regression is typically applied to predict probability of outcomes that are discrete (e.g. binary). Hence fitting $VC(t)$ directly instead of feeding in binary outputs may require some non-trivial manipulation.

10.4 Splines

Lines, whether we use them directly or put them in an exponent, are the fundamental building blocks of shape. After all, we can draw arbitrarily complex structures using solely combinations of lines. So if a relationship between variables is more complex than a single line can reflect, it is natural to simply throw more lines at the problem. If we take this to an extreme, every continuous function can be approximated by a sequence of line segments:

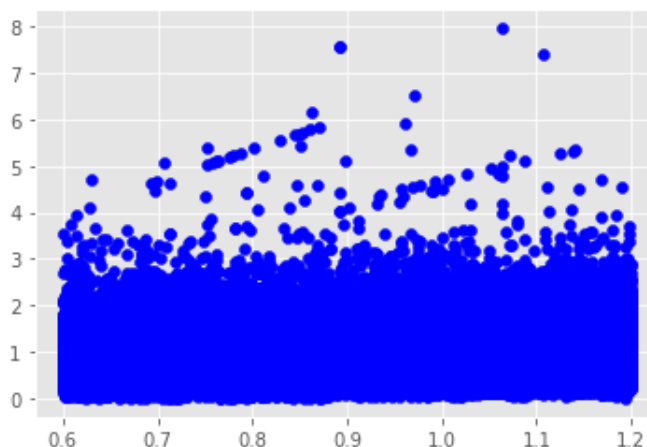


But allowing ourselves to use *too* many line segments to fit our data is likely to lead to over-fitting, like a lookup table approach with too many table entries. Splines are piecewise linear (or piecewise polynomial) functions that we can use to fit our data, setting parameters that encourage us to use fewer pieces when we can.



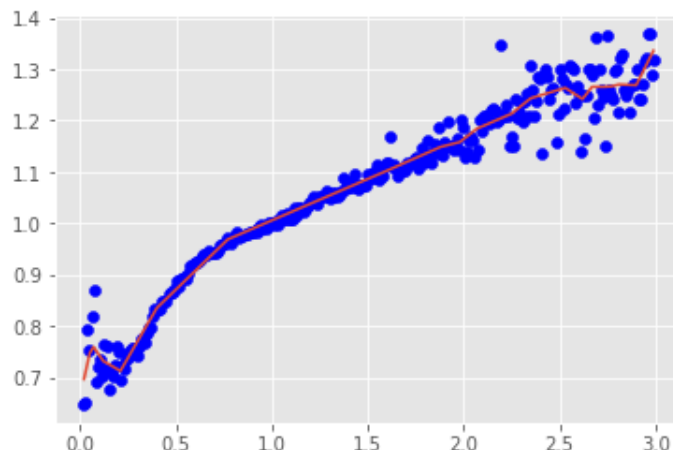
Let's apply splines to our volume curve prediction problem. We could do this in several different ways. For one, we could try to fit $VC(t)$ as a piecewise linear function of our features $HVC(t)$ and $\frac{V(t)}{ADV}$, rather than a linear function. To keep things simple though, let's reduce the dimension by dividing everything by $HVC(t)$. In other words, we'll try to fit $VC(t)/HVC(t)$ as a piecewise linear function of the single feature $\frac{V(t)}{HVC(t)*ADV}$.

We'll start with a raw look at what happens when we plot our target values of $VC(t)/HVC(t)$ on the y -axis and our feature values of $\frac{V(t)}{HVC(t)*ADV}$ on the x -axis for our training data. To get a better look at the meat of the data, we'll limit our x -axis range to be from 0.6 to 1.2, which roughly covers the middle half of the data points on this feature:



You wouldn't be blamed for asking what's the point of looking at plots like this? They look terrible! What am I expected to do with things like this? These are fair questions. But the point here is more cautionary than actionable. We should make plots like this, stare at plots like this, and get tattoos of plots like this (just kidding!) to internalize the fact that real financial data is often this much of a mess. Up to this point, we have been using happy looking data with nice distributions that satisfy the properties that give us a nice fit, but at least in finance such images are pretty unrealistic. So we should be careful of the extent to which they shape our thinking as we develop and test our predictive models.

That being said, there are some things we can do to make our real data more visually meaningful. Here, we'll round our feature values to two decimal places. Next, we'll compute a notional value-weighted average of all the target values that round to each two decimal feature value. We can graph these weighted averages of a wider range from 0.01 to 3 and we get something considerably less chaotic. It doesn't look like a line, but it does look like a reasonable curve that we can fit a linear spline to. Here is one such fit (with a somewhat ad hoc choice for the parameter that controls how closely the spline must fit the data):



If we allow the spline to deviate further from the data, it will use fewer linear pieces. If we force the spline to fit closer to the data, it will use more linear pieces. Finding a good balance between these things is not a topic we'll go into depth about here, but often you can just experiment with moving the fitting parameter up and down until you find a balance that serves your purposes. In this case, the spline consists of thirteen linear pieces.

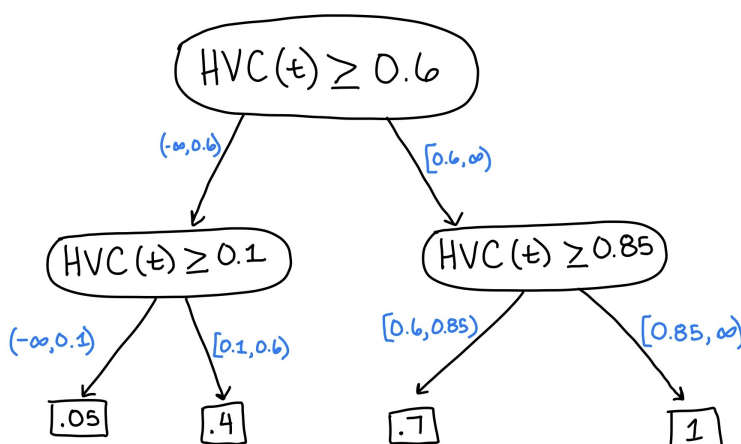
Let's see how this spline performs as a model for predicting the value $VC(t)$ on the test data. We'll apply this spline only when the feature value is in the range pictured above, and outside of that range, we'll just use $HVC(t)$ as our prediction for $VC(t)$. Compared to the baseline of always using $HVC(t)$ as the prediction, this approach yields 1.13 times the baseline in mean-squared error.

This poor performance might feel counter-intuitive, since our spline seems to fit the graph for the weighted average target values pretty well. Probably what this is telling us is that the variation around each average value in our graph is the main source of prediction error, and even a perfect fit of the averages is not going to remove that. Instead, we would need more and/or better features in our model to reduce variance.

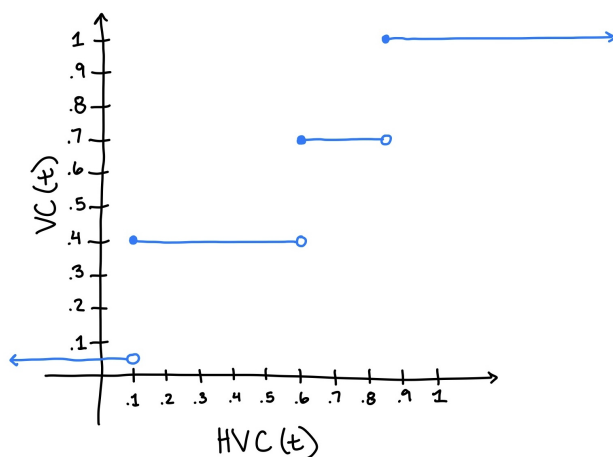
10.5 Decision Trees

Though they aren't often explained this way, *decision tree* models can be viewed as a basic subcase of splines. A decision tree encodes a set of binary tests that divide our data points into pieces, and then a constant function is applied on each piece. Since a constant function is the most basic form of line, this is essentially a degree 0 spline.

To see this, let's look at a toy example with just one variable. Let's use a small decision tree to predict $VC(t)$ based on the value of $HVC(t)$. Each internal node of the tree is marked with an inequality in terms of $HVC(t)$, and each leaf node is marked with a (constant) prediction for $VC(t)$. To evaluate the tree for a given value of $HVC(t)$, we start at the top and go to the left whenever the inequality is unsatisfied for our value, and go to the right whenever it is satisfied:

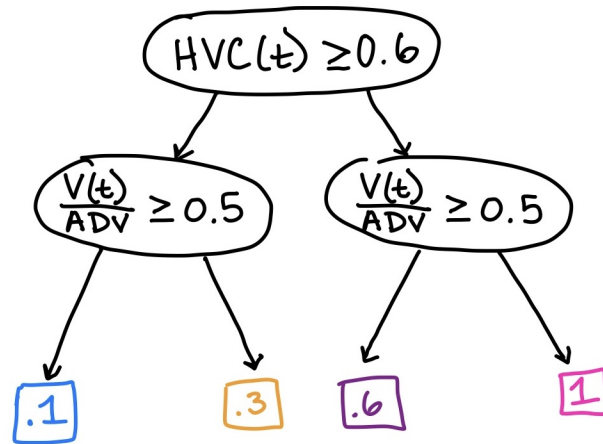


Here's an alternate way of visualizing the same thing:

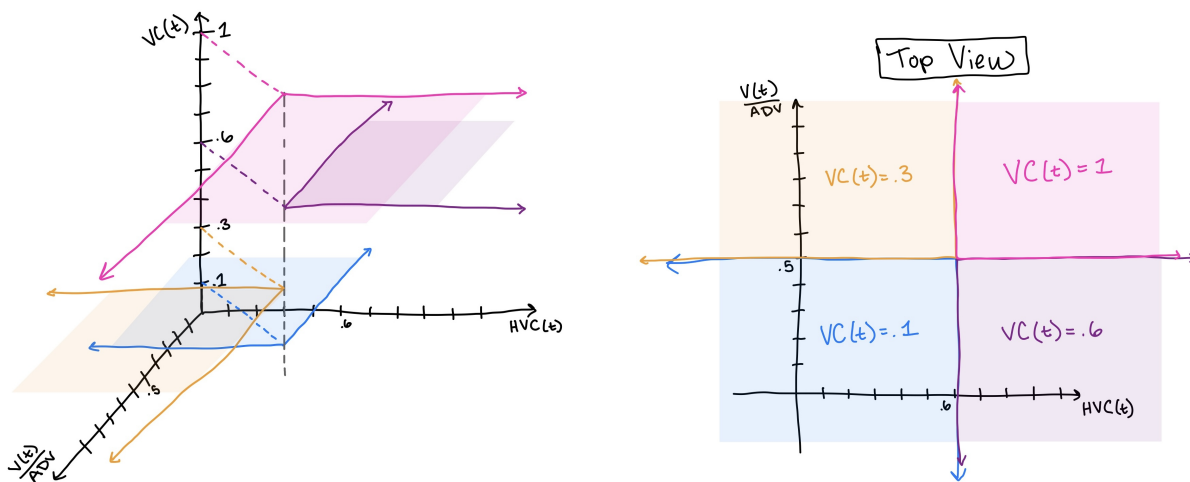


This step function view shows us that this is really a 0-degree spline, but without the constraint that pieces should connect at the knots. [Note that if we required connections at

the knots, then a 0-degree spline would just be a constant function.] Though it's easiest to look at things with a single variable, decision trees work similarly over an arbitrary number of variables. Here is a decision tree over two variables:



Here is a 3D plot of this decision tree (left) and a top-view of the same plot (right):

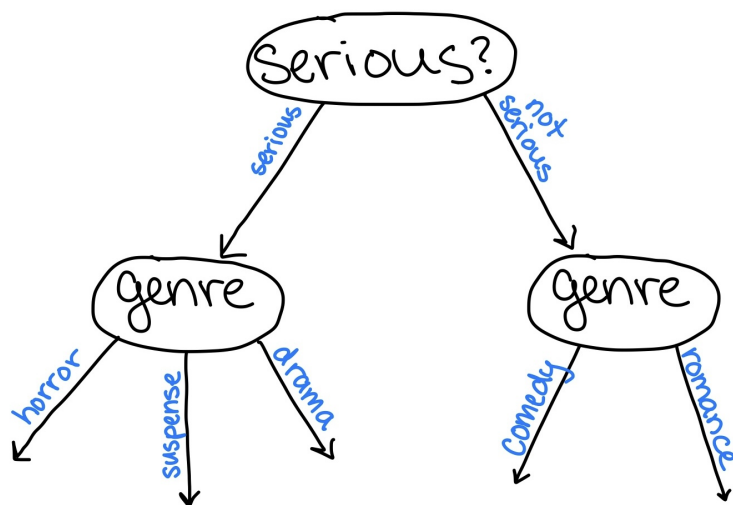


This same function could also be represented as a lookup table:

$HVC(t)$	$\frac{V(t)}{ADV}$	$VC(t)$
$(-\infty, 0.6)$	$(-\infty, .5)$	.1
$(-\infty, 0.6)$	$[.5, \infty)$	.3
$[0.6, \infty)$	$(-\infty, .5)$	.6
$[0.6, \infty)$	$[.5, \infty)$	1

In the extreme - if we allow a lot of rows in a lookup table, or a lot of leaves in a decision tree, or a lot of knots in a spline - all of these model forms can express the full set of all possible functions on a discrete and finite input space. So the differences between these model types for our applications are not fundamentally about expressibility, but rather about compression. In which form will it be easiest to find a well-fitting and well-performing model for our data? It often depends on the nature of our features and our data, which is why we might try all such forms with various degrees of freedom, even though they converge to the same set of all functions in the limit.

Decision trees will tend to fit data well when the data has a hierarchical structure. This matches with how decision trees can crop up in everyday life. For example, let's take the decision of what movie to watch. Here's a decision tree we might use for this task:



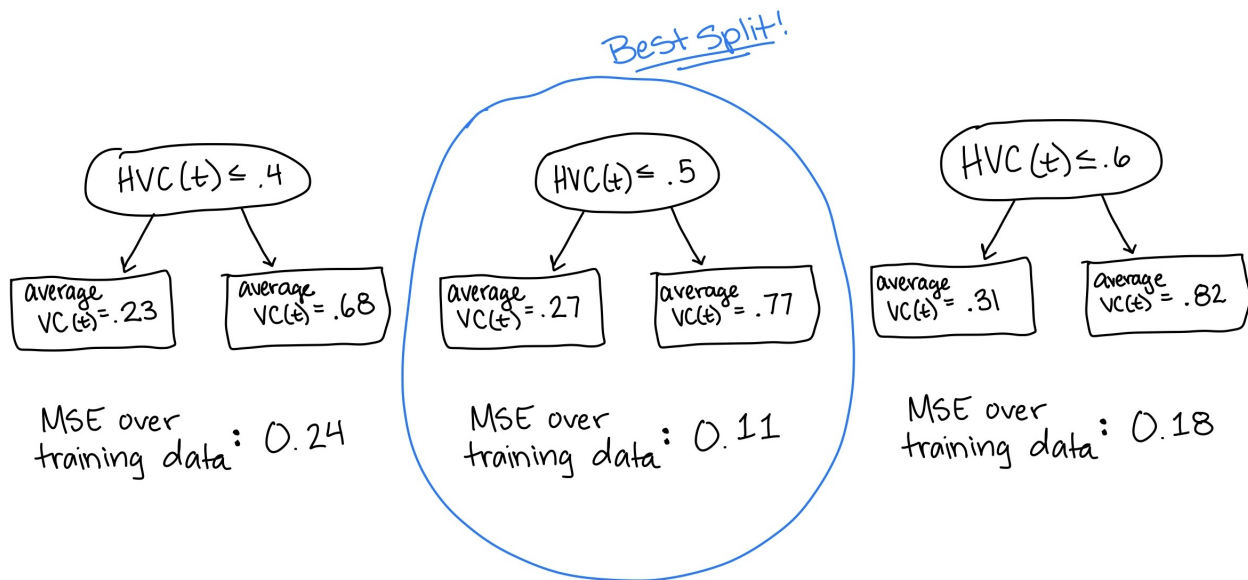
A top-level decision is serious vs. not-too-serious. Sometimes we're in the mood for a serious topic, and sometimes we are not. This decision definitely transcends the level of individual movies and rises to a first order principle in our decision making. At the next level for us are genre-based decisions: horror vs. suspense vs. drama, comedy vs. (god-forbid!) romance. This is the kind of decision process that a tree is good at capturing - hierarchical and layered.

Since model forms like decision trees can express arbitrarily complex functions in the limit, we need to mitigate the risk of over-fitting and control the computation time by capping the number of leaves we allow in our tree. For a given number of leaves, it is not known how to solve for the best-fitting tree for a particular training data set. But heuristic methods tend to find very good trees when they exist. It's not hard to come up with a decent greedy method, for example.

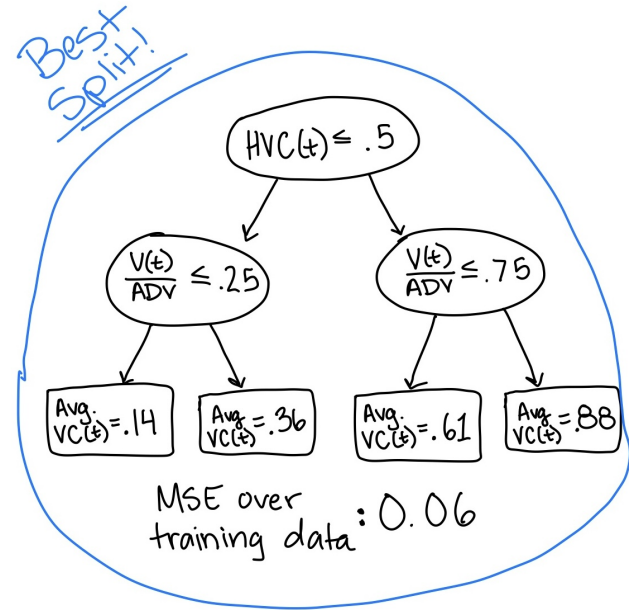
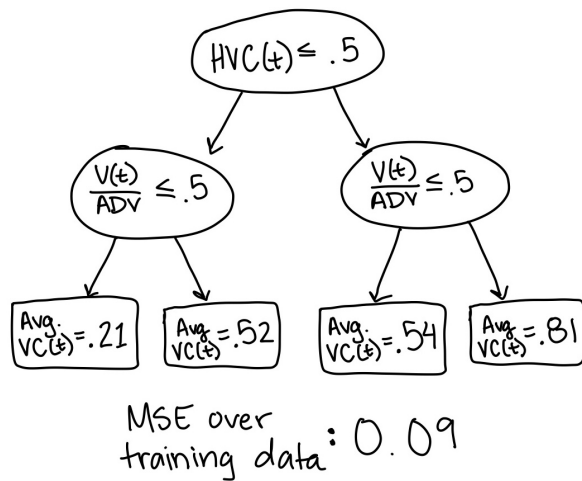
For our example of predicting $VC(t)$ from two features like $HVC(t)$ and $V(t)/ADV$, it could go roughly like this. We could start with a single node tree that produces a constant

prediction for $VC(t)$, namely the average value over training data. We could measure the quality of this model by looking at its mean-squared error over the training set. The astute reader might notice - this is precisely the variance of the value $VC(t)$ over the training set.

Now we will be interested in adding nodes to our tree one at a time, looking for a node to add that reduces the overall mean-squared error of our predictions by as much as possible. We could try a threshold based on $HVC(t)$ for example, and then label the resulting leaves with the average values of the training data points that fall on each side of the threshold. [Note: this example is not over real data, but is just giving a sense of what building a tree might look like.] We could compute the mean-squared error for this tree as a predictor of $VC(t)$ on our training data, and compare this to the mean-squared error of other thresholds we could choose:



Let's say we settle on the best of these as the first node in the tree. Then we will try to find a next layer of nodes that make a further dent in the overall mean-squared error, perhaps ending up with something like this:

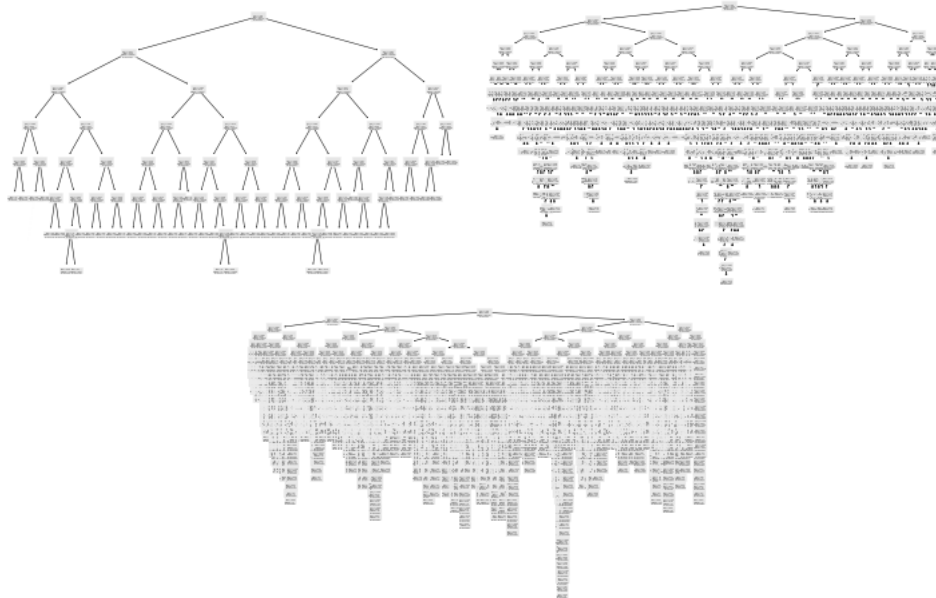


We can keep going, building a tree of whatever size we desire, trying to reduce the mean-squared error as much as we can with each additional node. The decision tree fitting algorithms that are built into machine learning software packages are more sophisticated than this, but our purpose here is just to provide a little intuition as to how such things can work.

This also illustrates why decision trees can be prone to overfitting (and why more leaves does not necessarily mean a better tree). Since we are using average $VC(t)$ of the training data for the leaf output, and we are evaluating MSE with respect to $VC(t)$ of the training data, with too many leaf nodes in the decision tree our model is just memorizing the average $VC(t)$ of the training data for very small groups of data points. This would give high accuracy on the training data, but would perform poorly when applied to a different testing data set.

Let's try building some decision trees on our real training data (for 2000 symbols during Q4 2021), and then seeing how they perform on our test data (same 2000 symbols on Q1 2022). We'll stick with the two features $HVC(t)$ and $V(t)/ADV$, and just vary the number of leaves we allow the tree to have. To get a general sense of how performance varies with the number of leaves, we'll try 50 leaves, 500 leaves, and 5000 leaves.

Here are what the respective trees look like:



It is fairly pointless to plot these when they have hundreds or thousands of leaves, but it can be mildly helpful for building our intuition about the structure of such models to force ourselves to look at the visual chaos from time to time. Unsurprisingly, a tree with only 5 leaves performs much worse than the baseline on our test data, producing roughly 1.87 times the mean-squared error. At 50 leaves, we're already beating the baseline, producing only about 0.83 times the mean-squared error of the baseline. The jump to 500 leaves gives us a small improvement, producing about 0.78 times the mean-squared error of the baseline. At 5000 leaves, the performance degrades slightly to about 0.80 times the baseline, suggesting that we have gone overboard and are beginning to overfit our model and chase noise in our training data that will only sour our predictive abilities.

There are a few related observations we can make in this instance. One is that our trees seem to perform much better than our linear regression model in a prior section, suggesting there is a fair amount of meaningful non-linearity here. Second, our trees can perform as well as a larger lookup table using only 500 leaves, which is perhaps surprisingly few. Our performance also doesn't seem that sensitive to the number of leaves we use, as we get similar performance in a wide range from somewhere less than 500 going all the way up to around 5000. It is important to note - using our test data performance to choose what size tree we want means that our test data isn't really neutral "test" data anymore. Instead, it's now more training data that is being used to fit a model parameter, namely the number of leaves. To get a true uncorrupted test, we should then take the final tree model and test it on completely fresh data that wasn't used to pick the tree size.

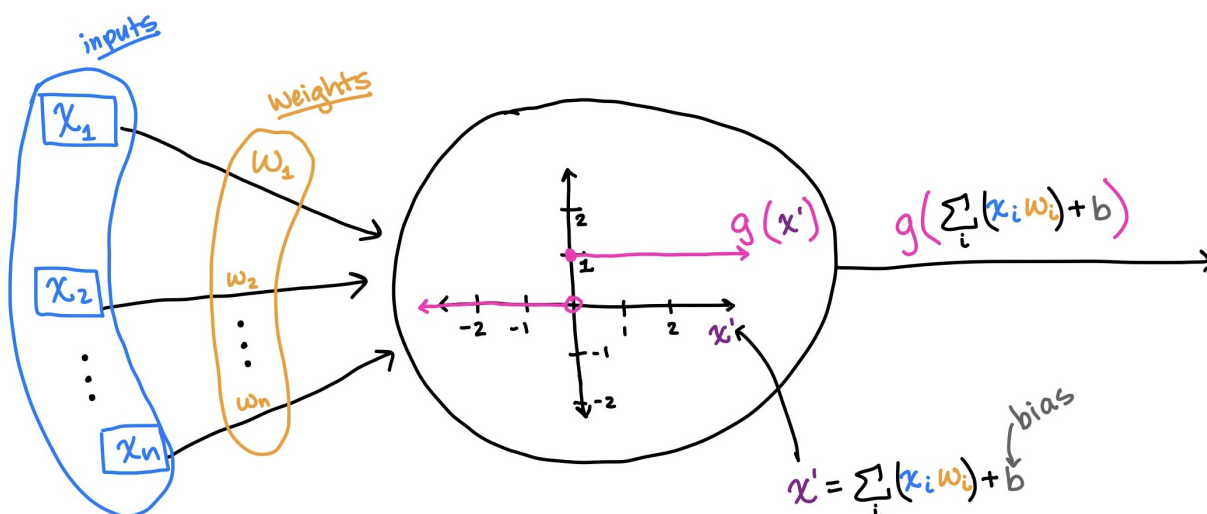
10.6 Neural Nets and Deep Learning

The first time Allison heard the phrase “deep learning,” she thought it was a good name for a grad student bar. She was a little disappointed that it turned out to be the branch of learning theory dealing with multi-layer neural nets. And don’t get us started on the name “neural nets,” which would be better as a name for a neuroscientist-only volleyball team.

But before we get to what neural nets and deep learning really mean, let’s start with the basic unit at their core: the perceptron. You might not recognize it all dressed up in today’s AI marketing hype, but the perceptron was a model introduced in 1943 by Warren McCulloch and Walter Pitts [12] and later extended by Rosenblatt [17] and Minsky and Papert [13]. It was originally meant as a simplified model of how neurons operate in the human brain. Since we’re way out of our depth on the biology here, we’re going to quote from the textbook, “Theoretical Neuroscience” by Abbott and Dayan [2] to explain:

“Neurons are highly specialized for generating electrical signals in response to chemical and other inputs, and transmitting them to other cells ... The elaborate branching structure of the dendritic tree allows a neuron to receive inputs from many other neurons through synaptic connections.” [pp. 3-4]

Got it? We don’t really, to be honest. But from a computer science standpoint, the important thing is: a neuron is a structure that can take in multiple inputs, combine and process them, and produce an output. The combination and processing is fairly simple in a perceptron model, and it looks like this:

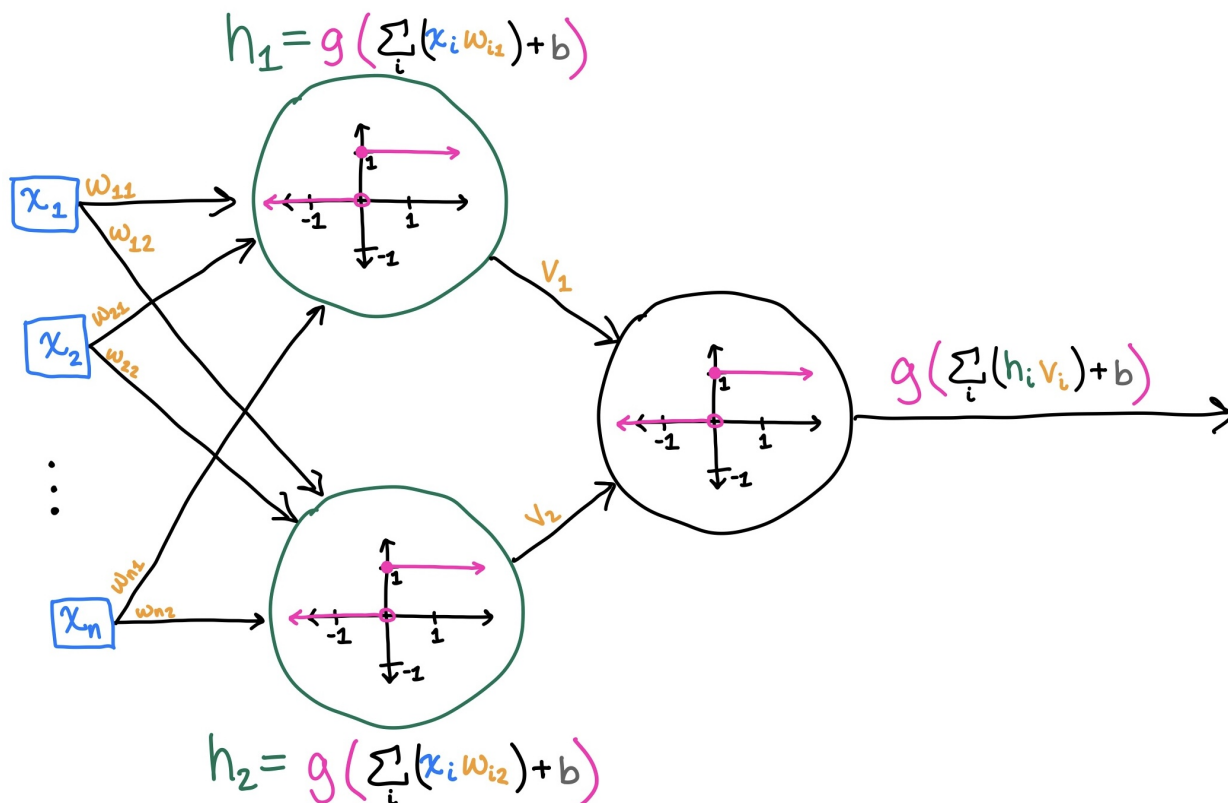


Here we are fixing weights for each input, computing a weighted sum, and then comparing the result to a threshold to produce a binary output. As a mathematical function, this can be written as:

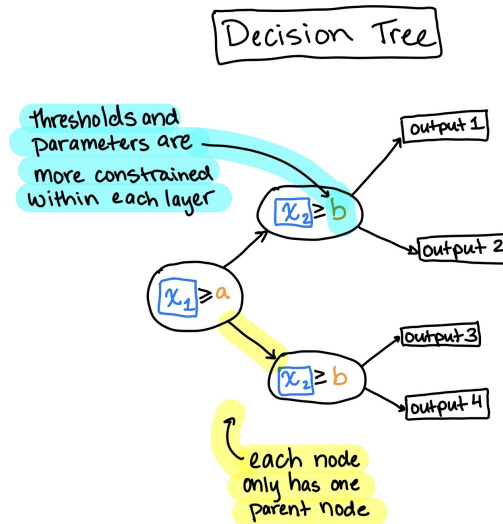
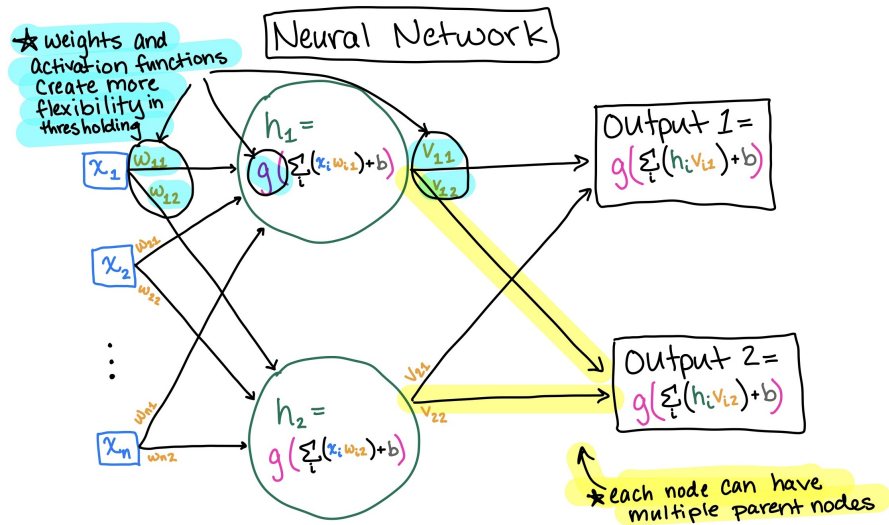
$$f(x_1, \dots, x_n) := \begin{cases} 1, & \sum_i w_i x_i + b \geq 0 \\ 0, & \sum_i w_i x_i + b < 0 \end{cases}$$

Which we can re-write for visual clarity as $g(\sum_i w_i x_i + b) = f(x_1, \dots, x_n)$.

We can also stack these little units as nodes in a directed graph. We'll keep things layered for now and have outputs for layer i serve as the inputs for layer $i + 1$. Here's what a small version might look like:



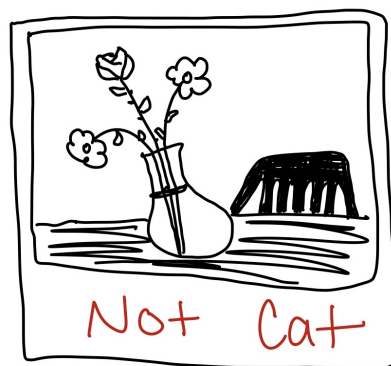
So far, this looks a lot like a decision tree, just drawn on its side. But there are a few important differences. First, taking weighted averages drastically expands the set of “features” that can be compared to thresholds at each node. Second, there is a reason we draw trees in vertical layers, as each node in the tree has exactly one parent in the layer above it. In neural networks that layer perceptrons together, there is no such restriction. What we have drawn above is a basic example of a *feedforward* network, where all edges go left to right, so there are no cycles. But each node can send its output to many nodes on the next layer, not just one. We can see these differences more clearly if we draw them side-by-side:



We may notice - these differences all make neural networks seem more powerful, as we have more degrees of freedom when constructing them. But we should remember - decision trees of arbitrary depth are already capable of expressing all boolean functions, and can closely approximate all reasonable functions over continuous domains and ranges. So these extra degrees of freedom are really only helpful if they align well with fitting the data we have to a more succinct model than a similarly well-fitting decision tree would require.

This raises a natural question: what kinds of data are neural nets particularly good at fitting and why? This question is at the center of highly active ongoing research, and we may never have a fully comprehensive answer, but there are some broad categories of things that are known. Neural nets are very effective at image classification, for example. It makes some intuitive sense why this should be. To see why, we can consider the classic conundrum of image classification: does this image contain a cat? If we stare at an image as humans, we tend to be pretty good at answering this question. We can decisively label the following

images as cat and not cat respectively:



But how can a computer, which “sees” an image as a grid of numbers representing the colors of pixels, determine such a label? After all, there is not a fixed pattern of pixels that solely represent a cat. A cat might appear in different positions in a picture, with different colors, at different scales, and in different orientations. It may also be partially obscured by other objects. We could imagine trying to make a long list of every pixel pattern that could represent a cat and searching brute force for one of these patterns in a particular image, but this seems like a hugely expensive and quixotic endeavor. Our list is unlikely to ever to be complete, and our search would get slower and slower as our list got longer and longer.

Instead, we’ll probably want to decompose the concept of a cat into a set of core sub-concepts. For instance, pointy ears, tail, and fur could be sub-concepts that we look for. The presence of such smaller patterns, as well as the relationships between them (e.g. do the pointy ears and the tail occur nearby each other and have similar scale?) can represent accumulating evidence towards the determination of “cat.” This kind of thinking maps well onto the way that model neurons accumulate input signals into outputs in a progression through a neural net. Though we shouldn’t get too carried away with this intuition. It is often incredibly difficult to tease out human-understandable sub-concepts from large, well-performing neural nets, and the extent to which this is even possible is a hot topic of research.

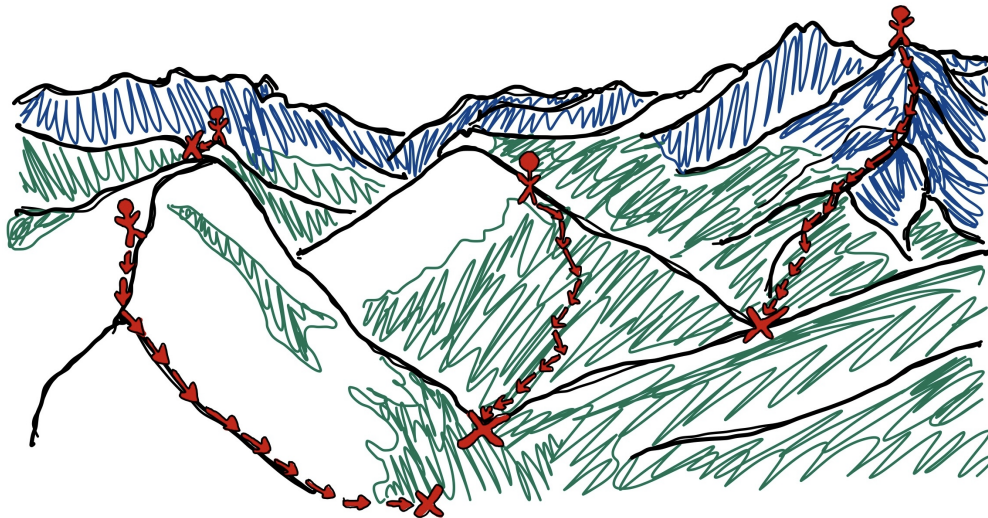
As you might infer from the increasingly convoluted image classification tasks that you encounter in CAPTCHAs, just *how good* neural nets are at image classification is a moving target, as people use newer techniques and better data sets for training. Currently, they are quite impressive, but still meaningfully worse than humans at many core tasks.

At a high level, the training process for neural nets works much like any training process in the realm of supervised machine learning. Ultimately, we are trying to find a neural net shape, a set of inputs, and a set of weight parameters that give “good” answers on our training set. We then test our trained models on fresh test data to see how well they do at new examples. Even if we limit ourselves to a fixed and reasonable number of model

neurons in our network, the set possible weight parameter combinations is much, much too large to search exhaustively, just like the set of all decision trees of a fixed depth becomes impractically large almost immediately. As in the case of decision trees, we will need heuristic methods for exploring the space of parameter settings effectively to find a “good” fit in an acceptable amount of time.

10.6.1 Gradient Descent and Sigmoid Functions

Our quantitative definition of a “good” fit can be in the same vein as what we used above for simpler models - we can search for parameters that bring down the mean-squared error of the predictions compared to the true answers in our training set. This kind of search can be done with a method called gradient descent, where we try making small adjustments to our parameters to see which directions of adjustments reduce our mean-squared error. One can imagine this process as trying to find a low valley while walking through a mountainous region in a thick fog. Suppose you can only see a few feet in front of you in any given direction. Which way would you walk? A natural approach is to take a step in the direction that is sloping downward, and then to re-evaluate. If your landscape does indeed look like somewhat smooth sloping mountains, and your step size is appropriate, you will get to a valley this way. If you try to approach from several different random starting points, you have a good chance of finding the lowest (or at least, a fairly low) valley with this strategy.



But what if your landscape does *not* look like gently sloping mountains, and instead looks like this:



In this kind of landscape, gradient descent is not going to work very well. The core problem is: small changes in the model parameters leading to big/chaotic changes in the overall error. This problem can arise particularly with threshold functions sitting in the individual nodes of our neural network. Threshold functions are brittle by construction - flipping suddenly from 0 to 1 when their inputs teeter on the edge of the specified threshold.

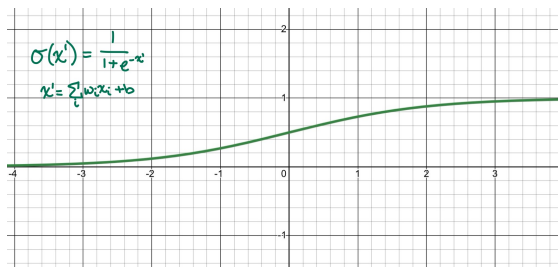
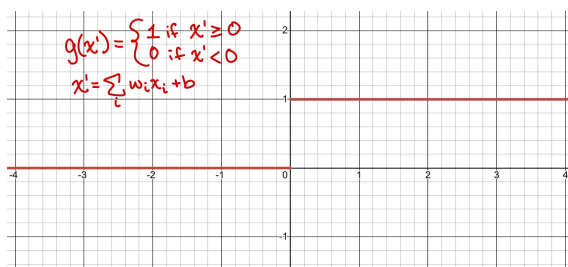
We can mitigate this potential issue by replacing threshold functions with something smoother. The sigmoid function, it turns out, is a good choice for this. Specifically, we'll replace our threshold function

$$f(x_1, \dots, x_n) := \begin{cases} 1, & \sum_i w_i x_i + b \geq 0 \\ 0, & \sum_i w_i x_i + b < 0 \end{cases}$$

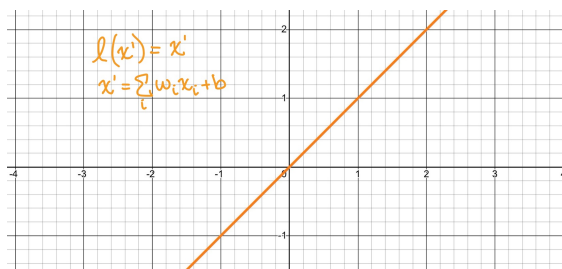
with a sigmoid function

$$\sigma(x_1, \dots, x_n) := \frac{1}{1 + e^{-(\sum_i w_i x_i + b)}}.$$

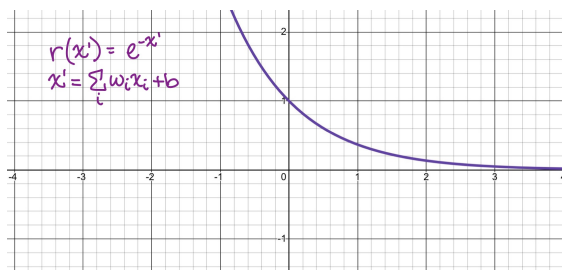
It's probably an understatement to say that this formula is not immediately intuitive. But it should seem familiar from our prior discussion on logistic regression. Let's graph these functions next to each other:



See how the sigmoid function is just a smoother version of the threshold function? As in the case of logistic regression, we can come up with this through the following process. We could start with just the line $y = x$:



Right now, both ends of this line go off to infinity. Negative infinity as x gets very negative, and positive infinity as x gets very positive. We ultimately want to get a function whose values are contained to 0 and 1, so we need some way to control an infinity and force it 0 or 1. We can do this by putting it in an exponent. Consider, for example, $y = e^{-x}$. Now, as x goes to negative infinity, y goes to positive infinity. But as x goes to positive infinity, y goes to 0. Thus we have succeeded in taming one side of the line:



Now we have a curve going up to positive infinity where we want to go 0 instead, and we have a curve going to 0 where we want to go to 1 instead. Luckily, the function $\frac{1}{1+y}$ maps very large positive values of y to very small positive value, and maps $y = 0$ to a value of 1. Composing this with what we have so far, we get the sigmoid function as a smooth approximation of the threshold function.

So if we assemble our neural networks out of nodes that employ sigmoid functions instead of threshold functions, the overall behavior will be a smoother function of our parameter choices, and this makes gradient descent more suitable as a method for finding a reasonably good fit to our training data. This is crucial, as searching through the entire space of possible values for the many weights becomes impractical nearly immediately as we add complexity and layers to our network shape.

The remaining thing we haven't covered yet is an explanation of *how* we can see a few feet ahead of us in every direction on our hypothetical mountain. Directions here represent

combinations of adjustments we could make to the various weight parameters, so a first idea might be to pick some small fixed adjustment δ and test every combination of $\pm\delta$ values as simultaneous adjustments to all weights. If we include the option to leave a weight as is, then every current weight value w could become $w - \delta$, w , or $w + \delta$. If we have N weight parameters, this would yield 3^N possible adjustment combinations to test. If we want to perform our tests on a rather ordinary computer, then something like 2^{30} tests is already quite ambitious, considering that each test may require re-computing performance over our whole training set. This corresponds to a value of $N = 30/\log_2(3)$, which is approximately $N \approx 19$. That is a pretty paltry number of weight parameters!

Luckily, there is a much better approach. To see why, let's do a quick thought experiment. Let's imagine that the mean-squared error of our model over our training set is a *linear* function of our weight parameters. Now, there are many things wrong that sentence. For one, a "squared" error is clearly not linear! And secondly, if error function *was* linear, it wouldn't have a minimum, as one end of the line would have to go off to negative infinity. But that's why we said "thought experiment" - just go with us for a moment and suspend disbelief.

Our linear (fake!) error function would look something like this:

$$Error = -0.2w_1 + 3.5w_2 + \cdots + 0.7w_{N-1} - 5.4w_N.$$

Here, the concrete numbers would come from pushing our actual training data through the neural network model as the original input x 's, and keeping track of all the weights as the variables $w_1, \dots, w_N$. Now, to compute which small $\pm\delta$ changes to the weights will jointly bring down our error as much as possible is easy! If w_i has a positive coefficient in our error function, we should decrease it by $-\delta$ to bring down our error. And if w_i has a negative coefficient, we should increase it by $+\delta$ to bring down our error. The linearity here means there is no meaningful interaction between the different weights, so we can compute the right adjustment for each individually, bringing our computation down to size N instead of size 3^N .

Of course, this is a fantasy. But - it is kind of true if we zoom in on our error function and squint hard enough! Remember what made splines so powerful? Every continuous curve can be approximated by little lines spliced together! If we make our adjustment size of δ small enough, then our more complex, definitely non-linear error function can be reasonably approximated by a linear function in a neighborhood big enough to cover our $\pm\delta$ steps. And we can simply compute the partial derivative for each weight parameter in the neighborhood of our current weight values in order to adjust it appropriately! That last sentence probably only made sense to you if you took and retained some amount of calculus, but no worries if

not. We're not going to dwell on this point much longer, so you can rejoin us in just a few more sentences.

The partial derivatives can be computed through a process called *back propagation*. This starts from the output of the model and works backwards. Computing the partial derivatives of the values going into the final output node is relatively easy, since they just pass through one last sigmoid function. Now let's step one layer back. The weight parameters on this layer affect the output indirectly, through the intermediation of the last layer. To compute their partial derivatives we can use the chain rule. Working our way back through the layers this way, we can figure out what small adjustments to the weights should lead to a net decrease in our cost function.

This was really a just a quick teaser on neural networks, gradient descent, and back propagation, all of which are big topics that many textbooks have been written about. If you want to do a deeper dive, you can find a more thorough but still beginner-friendly treatment in [15]. Applying neural nets to finance data is also a large and fascinating area of applied research. One thing to be wary of though is that additional degrees of modeling freedom always come with greater risk of unstable fitting, especially when noise/variance is high. For this reason, it is recommended that simpler models be tried first to establish improved baselines. If we are slow and deliberate about adding model complexity one bit at a time, we can choose a minimal model that achieves a particular level of performance, rather than, say, using neural net to do with thousands of parameters what we could do more reliably with a decision tree with hundreds of leaves.

Chapter 11

Closing thoughts

The contents of this textbook are by no means intended to represent a comprehensive or state of the art presentation of the material that is needed to be a successful quantitative researcher working with equities data. The goal here instead has been to provide a bit of a bird's eye view of the core scientific disciplines and techniques that may be relevant to modeling such financial data, as well as some of the common practical challenges of working with the levels of noise present in trading outcomes. We welcome feedback on these materials, as well as suggestions of further topics to cover in future educational materials.



Bibliography

- [1] Tversky A. and Kahneman D. Judgment under uncertainty: Heuristics and biases. *Science*, 185:1124–31, 1974.
- [2] Laurence F. Abbott and Peter Dayan. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. The M.I.T. Press, 2001.
- [3] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. *Theory of Computing [electronic only]*, 8, 01 2012.
- [4] T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld, and P. White. Testing random variables for independence and identity. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 442–451, 2001.
- [5] Avrim Blum. *On-Line Algorithms in Machine Learning*, pages 306–325. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.
- [6] Stephen Boyd. Ee364a - convex optimization i. <https://see.stanford.edu/Course/EE364A>.
- [7] Tamara Broderick. Machine learning (part 2). <https://www.youtube.com/watch?v=4baKKLtFK04>.
- [8] Tamara Broderick, Ryan Giordano, and Rachael Meager. An automatic finite-sample robustness metric: When can dropping a little data make a big difference? <https://arxiv.org/abs/2011.14999>, 2023.
- [9] Clément L. Canonne. *A Survey on Distribution Testing: Your Data is Big. But is it Blue?* Number 9 in Graduate Surveys. Theory of Computing Library, 2020.
- [10] Oded Goldreich and Dana Ron. *On testing expansion in bounded-degree graphs*, page 68–75. Springer-Verlag, Berlin, Heidelberg, 2011.

- [11] John P. A. Ioannidis. Why most published research findings are false. *PLOS Medicine*, 2(8), 08 2005.
- [12] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5, 1943.
- [13] Marvin Minsky and Seymour Papert. *Perceptrons*. M.I.T. Press, 1969.
- [14] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.
- [15] Michael A. Nielsen. Neural networks and deep learning. <http://neuralnetworksanddeeplearning.com/index.html>, 2015.
- [16] Karl Pearson. Vii. note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58(347-352):240–242, 12 1895.
- [17] F. Rosenblatt. The perceptron: A perceiving and recognizing automaton. 1957.
- [18] Paul Slovic, Melissa L. Finucane, Ellen Peters, and Donald G. MacGregor. The affect heuristic. *European Journal of Operational Research*, 177(3):1333–1352, 2007.
- [19] C Spearman. The proof and measurement of association between two things. *Am. J. Psychol*, 15:72–101, 1904.
- [20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.