On Equal footing

July 27, 2025

1 Introduction

Before we move ahead into more advanced discussions, this is a great time to get our vocabulary straight. Here, we will define a few terms that we will freely use in future notes.

2 How does a Machine-learning Problem differ from other more common problems that are solved using computers?

2.0.1 dterministic vs stochastic Problems

Think of some common problems that we use computers to solve - sorting a list of numbers in an ascending/descending order, finding the shortest path between 2 nodes in a graph, or maybe finding out whether a given number is prime or whether a given word is a palindrome. There are 2 common characteristics of *programs/procedures* that solve these kinds of problems:

- **Deterministic/algorithmic solution**: We can write down a step-by-step precedure to solve the problem. Another way of saying the same ting is that these problems have a deterministic algorithm that can solve them.
- **Provability**: Once we have laid out a step-by-step procedure to get to the desired solution we can even *prove* that the proposed algorithm is correct in the sense that it solves the problem. Sometimes we can additionally claim(and prove) that the algorithm we proposed is optimal. Or if not, we can provide some measure of sub-optimality of the algorithm.

And now think of some other common problems that we would like to use computers to solve - predict the stock prices of a certain company based on historical data,identify text from an image of someone's handwriting.Contrast these with the other examples given above, and you may notice :

- **Deterministic/algorithmic solution**: If you think hard enough you may catch yourself thinking- what does it even *mean* to solve for the

stock price?Isthere some formula for it based on the last-observed value(or maybe the last n observed values)?Or-what are the steps to decode a word or a digit from someone's handwriting ? Do we rely on some basic stroke-shapes? But different people have different handwriting ? How does our algorithm *generalize*. Note that this was not even an issue for deterministic algorithms. If you can prove that you can sort $\{1,\ldots n\}$ for any n, then your algorithm is genralized.

- **Provability**:Let's imagine we come up with some heuristic idea that we think will *solve* the problem(whatever that means), how do we prove that the algorithm is correct? What does *correctness* even mean in this context?

  If you really think about what makes these kinds of problems different from the nice,clean ones mentioned at the beginning we start to see some common factors -

  - There is some randomness in the very setup of the problem.This randomness could be because of the fluctuations in the stock market or the random variations in different people writing the same letter or digit.It is this randomness that makes it difficult to write a step-by-step procedure to solve these kinds of problems.
  - The randomness has a structure that creates a pattern(In general, there is no reason for this to hold true in the real world. Even the rsndomness may be random, and then we must throw our hands in the air).If we can deduce the pattern without having to write a gazillion if-else statements we may be able to *solve* this kind of a problem. For example, the market fluctuations may be best-described by some complicated function of factors - weather, time, who the current president is,etc.
  - We need to know what are all the so-called variables that affect the outcome. Now this is something completely new for us, in the sense that the weather or time or whether there is peace in the Middle East or not does not affect how we sort a bunch of numbers. But they most certainly do affect certain stock prices. So, to solve these kinds of problems, we also need to be able to know what influences the randomness. So, it turns out that to solve these kinds of problems, we need to *know* a lot of things - the pattern in the inherent randomness of the problem, what are all the factors that influence said randomness, etc. Whereas one-pointed,dedicated monastic study of one domain of problems may make us experts in answering these questions , but that will also imply a gap of a few decades between when someone asked us the problem and when we finally solved it. We can use computers to short-circuit the approach - Have a computer figure out the pattern in the randomness and what the most important factors influencing the ranodonmess are. That right there is the Learning in Machine Learning.

3 ML Problems

So, what makes a problem a *machine-learning* problem? There is no textbook answer to this, but once you've seen enough applications and types of ML problems hopefully this will be clear too. So, let's look at the types of Machine Learning Problems(broadly).

3.1 Types of problems

* **Predicting a value:** We've already seen an example of this class of problems - predicting stock prices of a certain company , predicting the weather(temperature,humidity,rainfall), housing prices in a fluctuating market, to name a few.
* **CLassifying an object**: We've also seen an example of this type of problem - classifying a letter or number from someone's handwriting, classifying photos of people, classifying email as spam/not spam , deciding whether a piece of text represents hateful/inciting speech(sentiment analysis).
* **playing(and winning) a game**: We haven;t seen examples of this class yet, but I'm sure we've heard of AlphaGo,DeepMind, etc. This class of problem asks a computer to learn a(possibly new) game and be good at it. When the game is unknown to the program, it is the computer's job to learn and formulate the optimal strategy.Imagine writing a deterministic program to do this. It is really difficult for some non-trivial games where we can find ourselves in one of a trillion trillion trillion possible states.

3.2 Mathematical formulation

: So far we've been talking about *solving* a machine-learning problem. But to do that we need to be able to *define* the problem. Let's now talk about how do go about defining a machine-learning problem mathematically. First, an assumption . We've seen how most machine-learning problems have an element of randomness inherent in them, and how we'd like a computer to *learn* the pattern in the randomness. To that end, we will make a simplifying assumption - That there exists a function $f(X1,\ldots, X_n)$ of some variables X_i (that we believe affect the problem, and that we can observe and measure)and that y=$f(X1,\ldots, X_n)$ is what we are trying to estimate/guess/calculate. Note that f can be any arbitrary function that need not even have a cosed form expression. We are happy if we can just somehow *learn* f. If you take a step back and really think about what we are trying to do, you may(*should*) have the following concern:

* Surely real-world phenomena are so random that we can't even hope to fully understand the randomness let alone capture it with just a function f. What gives ?!
  This is a valid concern. Since it may be difficult(or even impossible) to learn the actual randomness , what we *can* do is claim that the f is a reasonably good estimate of the actual randomness. This also gives us the advantage of interpretability(I wonder if that's a word). We may not have access to the true random process that generates the unknown variable but if we have access to the estimate f we can do some analysis on the random process and perhaps answer more general questions.This also helps us to provide an estimate on the correctness of our proposed algorithm using statistical analysis tools.

3.3 Jargon

Up until now I've been purposely verbose about a few things so that we know what we're talking about. Going ahead let's get familiar with some common terms - more as a chore that we do away with right now, so that we don't have to pore through lines of descriptive roundabout text that can be replaced with just one term. Although these may seem like mere definitions (which they most certainly are), but if you read through them, you may find that each of these terms is added here to introduce some other element of ML-problem solving or to link between some other concepts.

- **model**: A mahine-learning model(like any scientific model- Newton's law of motion, Newton's law of Gravitaion, Maxwell's equations of Electromagnetism) is our understanding of Nature's law (or some unknown law) represented mathematically. A model is used to calculate some unknown variable/quantity based on some observed variables/quantities. Referring to the preceding section, f is a model.
- **features**: The variables/quantities on which we believe influences our random process depends. These are the $X_i's$. These can be continuous or discrete- finite or infinite, scalar or vector.
- **target**: This is the unknown variable/quantity that we are interested to *solve for.*. This is the y we are estimating using the model. Again, this can be discrete or continuous, scalar or vector.
- **Training**: Recall a time when you took an exam. You may have gone through many exercises/practice problems a few(many?) days(nights?). That helped you *learn* some concept, and you were tested on that concept but on some new, unseen problem(data). You may have spent much more time training yourself on harder ubjects that required you to go through a lot of problems in order to understand a concept that was subtle or not so straightforward. In a similar way, a machine learning algorithm proceeds by training the computer on lots

of instances of the problem (that we supply to it-) called the *training data* with the hope that if the algorithm sees all possible patterns/variations in the data that can arise then it will be better at *learning* the pattern. This phase is called model training. Once the training is complete(more on How do we know when it's complete a bit later), we feed the model with unseen data to see how well it generalizes. In other words- how well has it understood the underlying pattern.
ASIDE: I have been using the word learning a lot by now and if it still seems confusing (What does learning really mean? Is the algorithm storing something? How does it remember what it has learnt?) we will see it clear as day when we get our hands dirty with the maths in a bit. Not unlike students, we can assigna score to model based on it's performance on test data. That's how we compare mahine-learning algorithms for the same task. Just like we are well prepared for the upcoming exam if we have seen (and solved) different types of problems, it is iportant(and the responsibility of the ML engineer) to train the model with training data that is not only representative of the kind of pattern that will be present in test data(We only train for the same syllabus as the exam quesrions), but we should also ensure that within the same kind of pattern, we have enough variation so that the model handle some unknown variation in the test phase. The above 2 requirements , when written mathematically imply that the training data and the test data better be sampled from the same distribution and that the training data better have some non-zero variance. This is done to make the model more robust.

- **overfitting**:Often the model may end up learning the pattern of the training data too well. It's like how we may remember how to solve exercise 4.2.a from Chapter 4 but we don't know how to solve an easy question on the exam. This kind of a model will not generalize well to unseen data.
- **underfitting**: This is the opposite. The model, much like a lazy student did not pick up *any* pattern in the training phase and hence it's predictive power is low.
- **Supervised learning**: These are situations when we show the model both the problem and the correct answer in the training phase so that it can correct itself while training.
- **Unsupervised learning**: There are some scenarios where we don't know the correct answer (or they're unavailable) so we let the model loose and let it figure out patterns on its own. A typical example would be to provide a bunch of points in some n-dimensional Euclidean space and ask the model to identify clusters (This is often done in social sciences graph analysis to separate clusters of similar people)