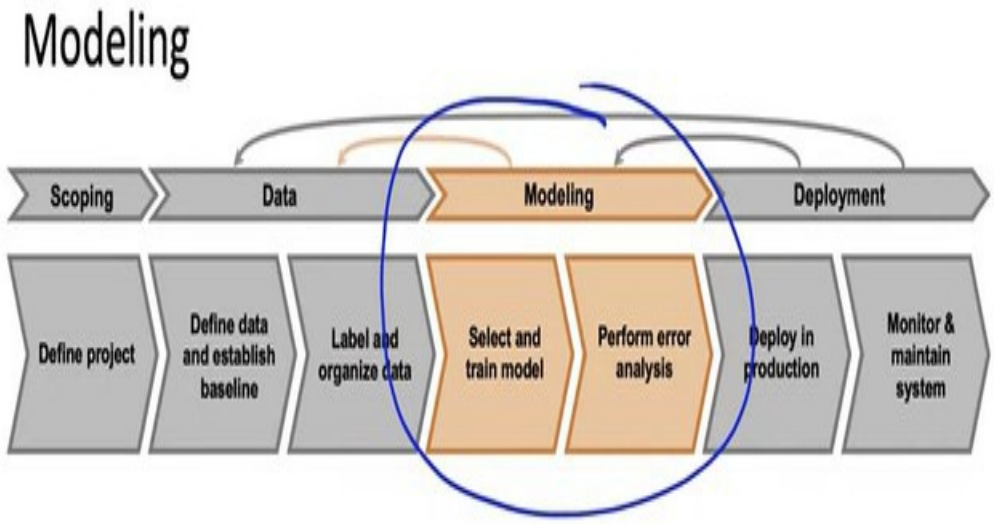
Introduction

In this section, we're going to focus on questions such as:

- What are the key challenges of trying to build a production-ready ML model?
- How do handle new datasets?
- What if you do well on the test set, but for some reason, that still isn't good enough for your actual application?



Model-centric AI development vs. Data-centric AI development

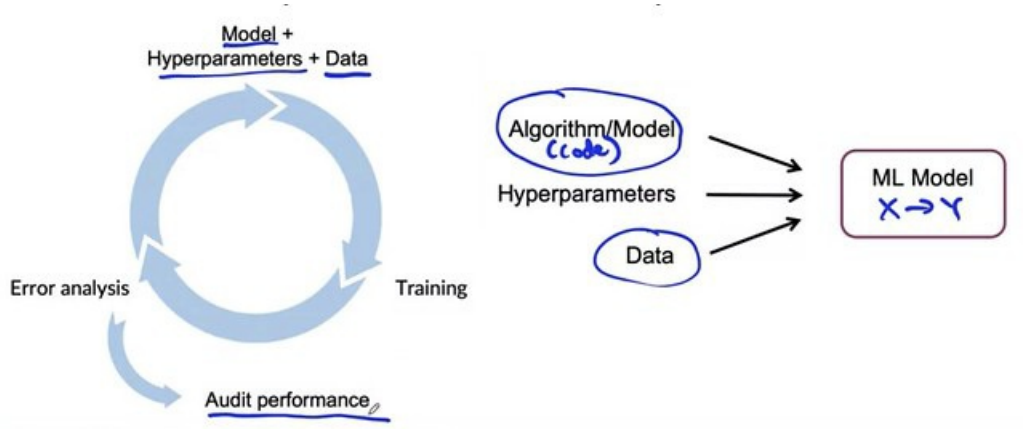
The way that AI has grown up, there's been a lot of emphasis on how to choose the right model (e.g. how to choose the NN architecture). For practical projects, it can be even more useful to take a more **data-centric approach**, where you focus not just on improving your ML model, but on making sure you're also ***feeding your algorithm high-quality data***.

Key Challenges

One framework to think about an AI system is: **AI system = Code + Data**

- There's been a lot of emphasis in the last several decades on how to improve the ML model. Researchers would just download a fixed dataset and fit the best model on it.
- But for a lot of applications, you have the flexibility to change the data if you don't like it.
    - There are a lot of projects where the model/algorithm is basically a solved problem. So, it'd be more efficient to spend time improving the data.

Model development is an iterative process



Because ML is such an empirical process, being able to go through this loop many times very quickly is key to improving performance.

After several iterations, it'd also be helpful to carry out a richer error analysis and do an audit to make sure it's working before you push it to production deployment.

Challenges in model development

When building a model, there are three key milestones that most projects should aspire to accomplish (order is important here).

1. Doing well on the training set (usually measured by averaging training error).
2. Doing well on the dev/test sets.
3. Doing well on business metrics/project goals.

Why low average error isn't good enough

As hard as it is to do good on the hold-out dataset, unfortunately, sometimes that's not enough. There are some other things needed to be done to make a project successful.

In addition to **data drift** and **concept drift**, there are some additional challenges we may have to address for a production ML project.

1. **Performance on disproportionately important examples:** A ML system may have a low average test error, but if its performance on a set of disproportionally important examples isn't good enough, then the ML system will still not be acceptable for production deployment.
    - **Example: Web search:** There are a lot of web search queries like these: "Apple pie recipe", "Latest movies", "Wireless data plan", "Diwali festival", etc. These types of queries are called *Informational and Transactional queries*. You just want to get some information about something you don't know much about. In such cases, you might be willing to forgive the search engine for

not giving you the **best** "apple pie recipe". There's a different type of queries such as "Standford", "Reddit", etc. which is called *Navigational queries*. Here, the user has a very clear intent to navigate to a website. So, they tend to be very unforgiving if a web search engine does anything rather than the right result (e.g. Standford --> Stanford.edu). **Navigational queries, in this case, are disproportionately important examples**.

- The challenge here is, of course, that **average test set accuracy tends to weigh all examples equally**.
- One thing you could do is to give disproportionately important examples a higher weight. That could work for some applications, but doesn't always solve the entire problem.

2. **Performance on key slices of the dataset (fairness):**
    - **Example: ML for loan approval:** Assume an ML system predicting who's going to repay a loan, and thus recommend approving certain loans for approval.
    - For such a system, you want to make sure it does not unfairly discriminate by ethnicity, gender, location, language, or other protected attributes.
    - Although the AI community was mostly had discussions about fairness in individuals, the fairness issue can also happen in other settings.
        - ***Example: Product recommendations from retailers:*** In recommendation systems of large retailers where you work with many vendors and brands, you want to be careful to treat fairly all major user, retailer, and product categories.
        - Even if an ML prediction system has a high average test set accuracy (i.e. it recommends better on average), if it gives very irrelevant recommendations to all users of one ethnicity, that may be unacceptable. OR if it always pushes products from large retailers and ignores smaller brands. OR the recommender never recommends a specific product category.

Performance on key slices of the dataset

Example: ML for loan approval

Make sure not to discriminate by ethnicity, gender, location, language or other protected attributes.

Example: Product recommendations from retailers

Be careful to treat fairly all major user, retailer, and product categories.

3. Rare classes: Specifically the cases of skewed data distributions.
    - Example: Medical diagnosis: In medical diagnosis, it's not uncommon for many patients not to have a certain disease, and therefore have a dataset where 99% of examples are negative and only 1% positive.
        - In such cases, you can achieve very good test set accuracy by writing a program that predicts "0" for everyone!
        - In medical fields, it's not acceptable to ignore (do not diagnose) ***obvious*** cases of illness.
        - This can also happen when only one (or a few) classes have very few observations. In such cases, even if you predict all the cases of the rare class wrong, you might still get high average test set accuracy.

Rare classes

Skewed data distribution

99% negative 1% positive

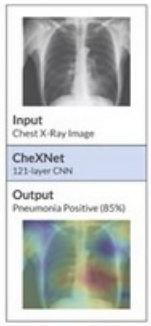
print("0")

Accuracy in rare classes

Condition	Performance
Effusion	0.901
Edema	0.924
Mass	0.909
Hernia	0.851

10,000 → Effusion

~100 → Hernia



Bottom line

> We need to go beyond just doing good on the test set.

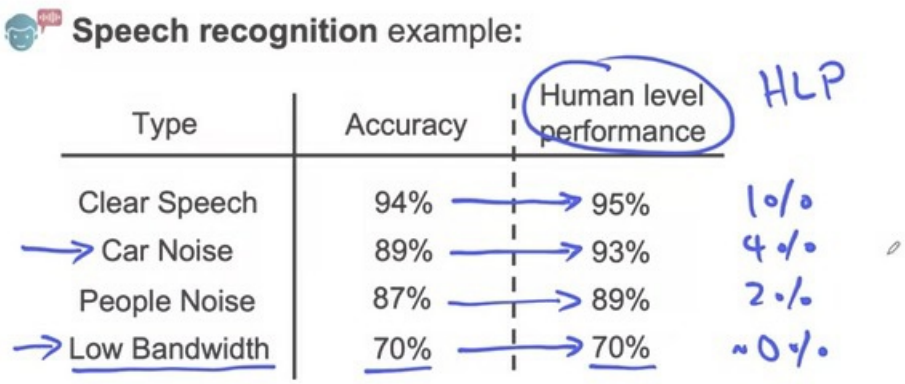
Establish a baseline

What are some of the best practices for quickly establishing a baseline?

Establishing a baseline level of performance

Let's assume for a speech recognition application, you've established these four major categories of speech:

- At first, you may say your model is not doing really god on "Low Bandwidth" because it has the lowest accuracy.
- But once you establish a baseline based on "human-level performance", you find out that the "Low Bandwidth" is actually doing best among the rest.



Unstructured and structured data

It turns out the best practices for establishing a baseline are quite different depending on whether you're working on unstructured or structured data.

Unstructured data tends to be data that humans are very good at interpreting. So, measuring human-level performance (HLP) is often a good way to establish a baseline.

In contrast, structured data are giant databases (e.g. sales transaction datasets), HLP is usually is a less useful baseline.

Ways to establish a baseline

- Human-level performance (HLP)
- Literature research for state-of-the-art/open source
- Quick-and-dirty implementation
- Performance of an older system

Baseline helps to indicate what might be possible. In some cases (such as HLP), also gives a sense of what is irreducible error/Bayes error.

By helping us to get a very rough sense of what might be possible, it can help us be much more efficient in terms of prioritizing what to work on.

Tips for getting started

Getting started on modeling

- Literature search to see what's possible (courses, blogs, open-source projects)
    - For practical applications (not research), don't obsses about the latest greatest algorithm. Instead, spend half a day (or a few days) reading and researching and then pick something reasonable that **lets you get started quickly**.
- Find open source implementations if possible.
- A "reasonable" algorithm with "good" data will often outperform a "great" algorithm with "not so good" data.
- The point is to get started on the iterative process of an ML system. You don't want to spend a lot of time just picking the model.

Deployment constraints when picking a model

Should you take into account deployment constraints (e.g. compute constraints) when picking a model?

Yes, if baseline is already established and goal is to build and deploy. ***No*** (or not necessarily), if purpose is to establish a baseline and determine what's possible and might be worth pursuing.

Sanity-check for code and algorithm

- Try to overfit a small training dataset before training on a large one. It helps you find bugs much more quickly.
    - Example 1: Speech recognition: X (audio) --> y (transcript), just train on one example just to see if the output makes sense.
    - Example 2: Image segmentation: before training hours many examples, just feed it one example to see if it can at least overfit one training example.