Lecture 2: Introduction to Machine Learning Systems Design [Draft]

CS 329S: Machine Learning Systems Design (cs329s.stanford.edu)

Prepared by Chip Huyen & the CS 329S course staff
Reviewed by Andrey Kurenkov, Luke Metz, Laurens Geffert
Errata and feedback: please send to chip@huyenchip.com

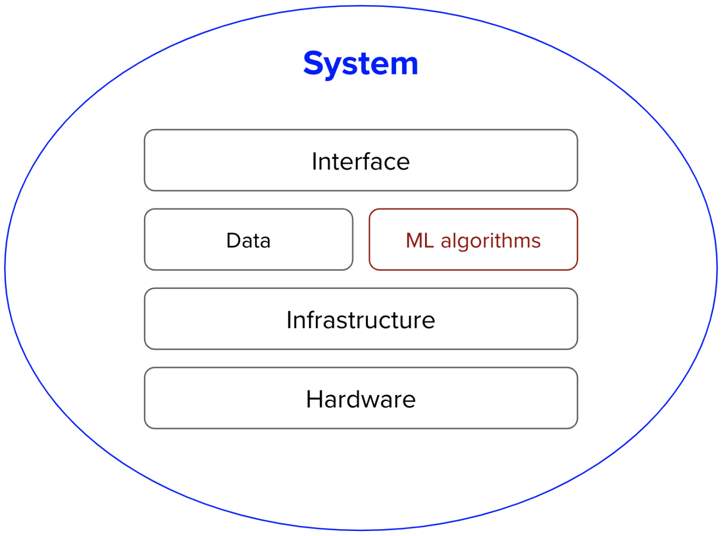
Note:

1. See the course overview and prerequisites on the first lecture slides.
2. The course, including lecture slides and notes, is a work in progress. This is the first time the course is offered and the subject of ML systems design is fairly new, so we (Chip + the course staff) are all learning too. We appreciate your:
    a. **enthusiasm** for trying out new things
    b. **patience** bearing with things that don't quite work
    c. **feedback** to improve the course.

1. Goals of machine learning systems design

Machine learning systems design is the process of defining the **interface, algorithms**[1]**, data**, **infrastructure**, and **hardware** for a machine learning system to satisfy **specified requirements**.

Most ML courses only cover the ML algorithms part. In this course, we won't teach you different ML algorithms, but we'll look at the entire system.



Requirements vary from use case to use case. Here are the four main requirements that the systems we'll be studying should have.

1. Reliable
2. Scalable
3. Maintainable
4. Adaptable

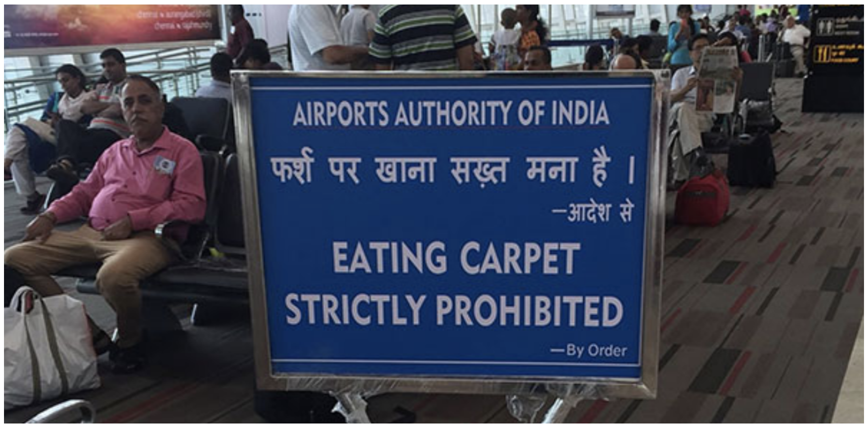
Reliability

The system should continue to perform the **correct function** at the **desired level of performance** even in the face of **adversity** (hardware or software faults, and even human error).

"Correctness" might be difficult to determine for ML systems. For example, your system might call the function ".predict()" correctly, but the predictions are wrong. How do we know if a prediction is wrong if we don't have ground truth labels to compare it with?

[1] Algorithms include both machine learning algorithms and other algorithms.

When traditional software systems, you often get a warning, e.g. systems crash or runtime error or 404. However, ML systems fail silently. Users don't even know that the system has failed and might have kept on using it as if it was working.



An example of possibly a silent fail of Google Translate

Scalability

As the system grows (in data volume, traffic volume, or complexity), there should be reasonable ways of dealing with that growth.

Scaling isn't just scaling up -- expanding the resources to handle growth. In ML, it's also important to scale down -- reducing the resources when not needed. For example, at peak, your system might require 100 GPUs. However, most of the time, your system needs only 10 GPUs. Keeping 100 GPUs up all the time can be costly, so your system should be able to scale down to 10 GPUs.

An indispensable feature in many cloud services is autoscaling: automatically scale up and down the number of machines depending on usage. This feature can be tricky to implement. Even Amazon fell victim to this when their autoscaling feature failed on Prime Day, causing their system to crash. An hour downtime was estimated to cost it between $72 million and $99 million[2].

Maintainability

There are many people who will work on an ML system. They are ML engineers, DevOps engineers, subject matter experts (SMEs). They might come from very different backgrounds, with very different languages and tools, and they should all be able to work on the system productively.

[2] https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7

Importance of subject matter experts

Subject matter experts (doctors, lawyers, bankers, farmers, stylists, etc.) are not only users but also developers of ML systems. SMEs are often overlooked in the design of ML systems, but many ML systems wouldn't work without subject matter expertise.

Most people only think of subject matter expertise during the data labeling phase -- e.g. you'd need trained professionals to label whether a CT scan of a lung shows signs of cancer. However, an ML system would benefit a lot to have SMEs involved a lot more steps, such as:

- problem formulation
- feature engineering
- error analysis
- model evaluation
- reranking predictions
- user interface: how to best present results to users and or to other parts of the system

Collaboration among cross-functional teams

There are many challenges that arise from having multiple different profiles working on a project. For example, how do you explain ML algorithms limitations and capacities to SMEs who might not have engineering or statistical backgrounds? To build an ML system, we want everything to be versioned, but how do you translate domain expertise (e.g. if there's a small dot in this region between X and Y then it might be a sign of cancer) into code and version that? Good luck trying to get your doctor to use Git.

To help SMEs get more involved in the development of ML systems, many companies are building no-code/low-code platforms that allow people to make changes without writing code.

Adaptability

To adapt to changing data distributions and business requirements, the system should have some capacity for both discovering aspects for performance improvement and allowing updates without service interruption.

Because ML systems are part code, part data, and data can change quickly, ML systems need to be able to evolve quickly. This is tightly linked to maintainability. We'll go more into this later!

2. Different types of machine learning systems

Before we start designing an ML system, let's take a look at the existing machine learning systems and see if we can find some patterns. We'll be categorize different systems based on three aspects:

- How their ML models serve their predictions (batch prediction vs. online prediction)

- Where the majority of computation is done (edge computing vs. cloud computing)
- How often their ML models get updated (online learning vs. offline learning)

This section provides an overview of these types of systems. We'll be going deeper into how to implement them in the future in lectures about deployment.

Caution against categorical thinking

This categorization provides an anchor to think about the requirements and limitations of different systems. It's not a guide. Seemingly different ways of doing things might be fundamentally similar, and the choices don't have to be mutually exclusive. For example, you don't have to do only batch predictions or only online predictions -- you can do both. If you're doing batch predictions, switching to online predictions might be easier than you think. You shouldn't think about whether to do batch predictions or online predictions, but how best to address your problem. Focus on solutions, not techniques.

Putting things in buckets might cause organizations to get stuck in one way of doing things without realizing that the other way isn't that different but can provide much more value. If you're interested in learning more about the dark side of "categorical thinking", check out this great HBR psychology article[3].

Online prediction vs. batch prediction

If you've ever used the AI platforms offered by popular cloud providers like AWS or Google Cloud, you'll see that they offer two options for making predictions: **batch prediction** and **online prediction** (sometimes called **HTTP prediction**).

⚠ **Misnomer warning** ⚠

The terms **online prediction** and **batch prediction** can be confusing. Both can make predictions for multiple samples (in batch) or one sample at a time.

I suspect the reason companies don't use **online prediction** vs. **offline prediction** is because if you put your models on the cloud, then predictions are technically made "online" - over the Internet.

- **Batch prediction** is when predictions are generated asynchronously and periodically (e.g. every four hours or whenever triggered). The predictions are generally stored somewhere such as in SQL tables or CSV files and retrieved as needed. Batch prediction

[3] Thanks ZhenZhong Xu for sharing this!

is good for when you don't need the results immediately, but want to process a high volume of samples.

  Batch prediction can also be used as a trick to reduce latency for more complex models. Generating predictions on the spot for each incoming query might take too long, but retrieving a precomputed query might be faster.

  One common use case of batch prediction is recommendation systems -- generating recommendations for users every few hours and only pull out the recommendation for each user when they log into the system.

  You don't have to use all the predictions generated. For example, you can make predictions for all customers on how likely they are to buy a new product, and reach out to the top 10%.

- **Online prediction** is when predictions are generated as requests arrive and returned as responses. Online prediction is necessary when you need the prediction immediately for each data sample, e.g. you want to know whether a transaction is fraudulent as soon as it happens.

Online prediction and batch prediction don't have to be mutually exclusive. One hybrid solution is that you do online prediction for default, but for common queries, predictions are precomputed in advance and pulled out as needed to reduce latency when it takes longer to make a prediction than to retrieve a result.

In many applications, online prediction and batch prediction are used side by side for different use cases. For example, food ordering apps like DoorDash and UberEats use batch prediction to generate restaurant recommendations -- it'd take too long to generate these recommendations online because there are many restaurants. However, once you click on a restaurant, food item recommendations are generated using online prediction.

Batch prediction is a workaround for when online prediction isn't cheap enough or isn't fast enough (or legacy systems force you to use batch prediction because it'd be too expensive to update your infrastructure to online prediction). Why generate 1 million predictions in advance and worrying about storing and retrieving them if you can generate each prediction as needed at the exact same cost and same speed? Online prediction can also allow many use cases previously impossible with batch prediction.

As hardware becomes more customized/powerful, better techniques are developed to allow faster, cheaper online predictions, it will make sense to switch to online prediction.

	Batch prediction	**Online prediction**
Frequency	Periodical (e.g. every 4 hours)	As soon as requests come
Useful for	Processing accumulated data when you don't need immediate results (e.g. recommendation systems)	When predictions are needed as soon as data sample is generated (e.g. fraud detection)
Optimized	High throughput	Low latency
Input space	Finite: need to know how many predictions to generate	Can be infinite
Examples	Netflix recommendations	Google Assistant speech recognition

Edge computing vs. cloud computing

ML models are compute-intensive. If the devices where the ML models are used aren't powerful enough to handle the needed computation, the computation has to be done on a server.

Edge computing is when the large chunk of computation is done on the edge (browsers, phones, laptops, smart watches, cars, security cameras, robots), whereas cloud computing is when the large chunk of computation is done on the cloud (hosted servers).

For cloud computing, you'd need network connections to be fast enough to transfer data between devices and servers.

For edge computing, you'd need edge devices that:

- are powerful enough to handle the computation
- have enough memory to store ML models and load them into memory
- have enough battery or be connected to an energy source to power the application for a reasonable amount of time (running a full-sized BERT on your phone, if your phone is capable of doing that, is a very quick way to kill its battery).

Benefits of machine learning on the edge

Edge computing has become a buzzword in the recent year for a reason. There are many benefits for ML systems to be on the edge.

1. **Can work without (Internet) connections or with unreliable connections**
   Cloud computing relies on stable connections. It wouldn't work for situations where there is no connection (e.g. many companies have strict no-Internet policy because of privacy concerns) or connections are unreliable (e.g. rural areas or developing countries).

Caveat: edge computing means that you don't need connections to execute computation, but your systems might still need connections to function. For example, ETA estimation needs Internet connections to receive external traffic information to provide accurate estimations.

2. **Don't have to worry about network latency**
   Requiring data transfer over network might make some use cases impossible because of network latency. In many cases, network latency is a bigger bottleneck than inference latency.
3. **Fewer concerns about privacy**
   ML on the edge means that your systems will less likely have to send user data over networks, which can be intercepted. Cloud computing often means storing data of many users in the same place, which means a breach can affect many people[4]. Edge computing also makes it easier to comply with regulations (e.g. GDPR) about how user data can be transferred or stored.

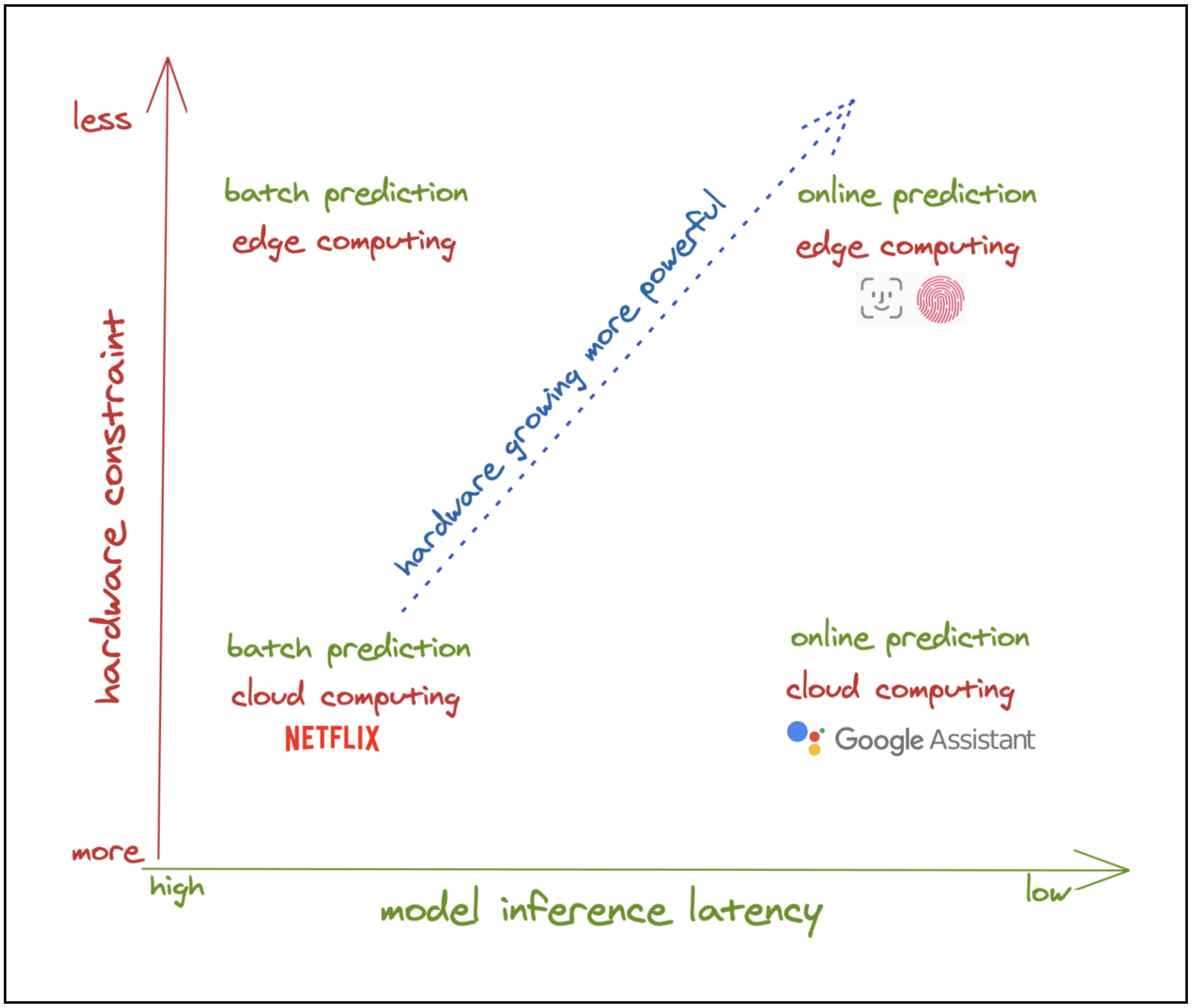
   Caveat: edge computing doesn't mean no privacy concerns. In some cases, edge computing might make it easier for attackers to steal user data, e.g. they can just take the device with them!
4. **Save on server cost**
   The more computations we can push to the edge, the less servers we'll need, and the less we'll have to pay for servers.

Because of the many benefits of edge computing over cloud computing, companies, both startups and large companies, are in a race to develop more powerful hardware optimized for different ML use cases. We'll dig deeper into edge devices and edge computing in future lectures.

Future of ML: online and on-device

As hardware becomes more powerful and optimized for ML, I believe that ML systems will transition to making online prediction on-device. In addition to making predictions on-device, people are also working on techniques that enable ML model training over edge devices (e.g. federated learning[5]), and we'll go over this in the future lectures.

[4] Nearly 80% of Companies Experienced a Cloud Data Breach in Past 18 Months (Security magazine, 2020)
[5] Federated Learning: Collaborative Machine Learning without Centralized Training Data



Online learning vs. offline learning

Online learning is a really exciting but underexplored area of ML. It's also hard to do so few companies are actually doing it, but I believe more and more companies will do it in the future. Below are a few differences between online learning and the traditional offline learning that most people are more familiar with. Online learning requires different infrastructure and some mental shift, which (again) we will cover in the future.

	Offline learning	**Online learning**
Iteration cycle	Periodical (months)	Continual (minutes) *This is different from continuous -- learning with every coming sample!*
Batch size	Thousands to millions	Hundreds

Data usage	Each sample seen multiple times (epochs)	Each sample seen at most once
Evaluation	Mostly offline evaluation	Offline evaluation as sanity check Mostly relying on online evaluation (A/B testing)

Online learning doesn't mean offline learning. The companies that have most successfully used online learning also train their models offline in parallel and then combine the online version with the offline version.

If the infrastructure is set up right, there's no fundamental difference between online learning and offline learning, just a hyperparameter to tune, e.g. how big is the (micro)batch, how often to evaluate your model.

3. Iterative process

Before deploying my first ML system, I thought it'd go like this.

1. Collect data
2. Train model
3. Deploy model
4. Profit

But in reality, it went something like this. Here is one common workflow that you might encounter when building an ML model to predict whether an ad should be shown when users enter a search query[6].

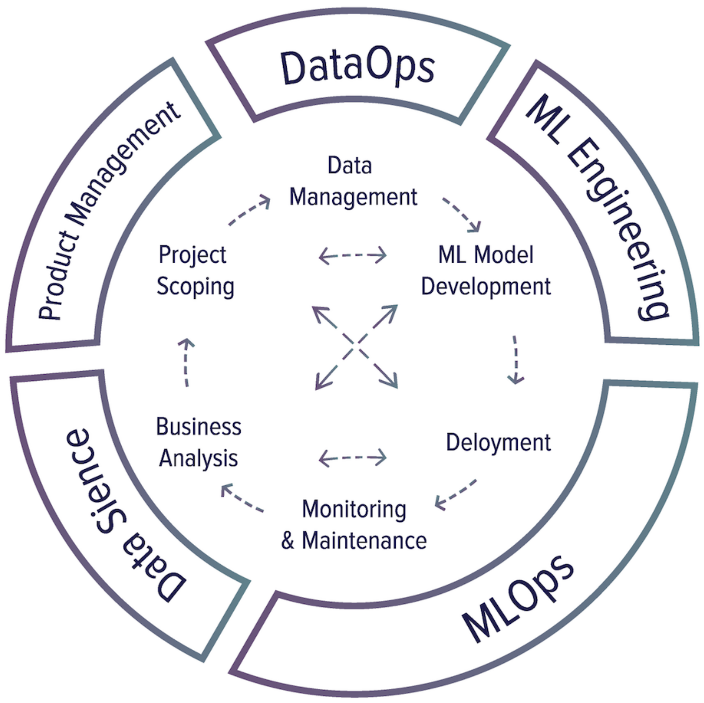
1. Choose a metric to optimize. For example, you might want to optimize for impressions -- the number of times an ad is shown.
2. Collect data and obtain labels.
3. Engineer features.
4. Train models.
5. During error analysis, you realize that errors are caused by wrong labels, so you relabel data.
6. Train model again.
7. During error analysis, you realize that your model always predicts that an ad shouldn't be shown, and the reason is because 99.99% of the data you have is no-show (an ad

[6] Praying and crying not featured but present through the entire process.

shouldn't be shown for most queries). So you have to collect more data of ads that should be shown.

8. Train model again.
9. Model performs well on your existing test data, which is by now two months ago. But it performs poorly on the test data from yesterday. Your model has degraded, so you need to collect more recent data.
10. Train model again.
11. Deploy model.
12. Model seems to be performing well but then the business people come knocking on your door asking why the revenue is decreasing. It turns out the ads are being shown but few people click on them. So you want to change your model to optimize for clickthrough rate instead.
13. Start over.

There is a lot of back and forth between different steps, making the process more of a cycle than a waterfall. There are six main steps in this cycle.



1. **Project scoping**
   A project starts with scoping the project, laying out goals & objectives, constraints, and evaluation criteria. Stakeholders should be identified and involved. Resources should be estimated and allocated.
2. **Data management**

Data used and generated by ML systems can be large and diverse, which requires scalable infrastructure to process and access it fast and reliably. Data management covers data processing, data control, data storage, and data consumer.

3. **ML model development**
   From raw data, you need to create training datasets and possibly label them, then generate features, train models, optimize models, and evaluate them. This is the stage that requires the most ML knowledge and is most often covered in ML courses.
4. **Deployment**
   After a model is developed, it needs to be made accessible to users.
5. **Monitoring and maintenance**
   Once in production, models need to be monitored for performance decay and maintained to be adaptive to changing environments and changing requirements.
6. **Business analysis**
   Model performance needs to be evaluated against business goals and analyzed to generate business insights. These insights can then be used to eliminate unproductive projects or scope out new projects.

4. Project scoping

Goals and objectives

Goals

While most companies want to convince you otherwise, the sole purpose of businesses, according to the Nobel-winning economist Milton Friedman, is to maximize profits for shareholders[7]. The ultimate goal of an ML project or any project within a business is, therefore, to increase profits.

Aside: non-profits are, of course, exceptions from this rule. There are a lot of exciting applications of ML/AI for social good.

- environment (climate change, deforestation, flood risk, etc.)
- public health
- education (intelligent tutoring system, personalized learning)

ML projects might have goals that directly or indirectly improve the bottom line. Directly such as increasing sales (conversion rates), cutting costs. Indirectly such as higher customer satisfaction, increasing time spent on a website.

[7] A Friedman doctrine-- The Social Responsibility Of Business Is to Increase Its Profits (Milton Friedman, The New York Times Magazine 1970).

The financial effect of an indirect goal can be confusing. An ML model that gives customers more personalized solutions can make them happier which makes them spend more money on your services. The same ML model can also solve their problems faster which makes them spend less money on your services.

Example: when building a ranking system for a newsfeed, some of the possible goals are:

1. minimize the spread of misinformation
2. maximize revenue from sponsored content
3. maximize users' engagement

Objectives

Goals define the general purpose of a project. Objectives define specific steps on how to realize that purpose. For example, the goal of a ranking system for a newsfeed is to maximize users' engagement and the objectives are:

1. Filter out spam
2. Filter out NSFW content
3. Rank posts by engagement: how likely users will click on it

However, because of the questionable ethics of optimizing for engagement (extreme posts get more engagements -> promotions of more extreme ideas[8][9]), we want to create a more wholesome newsfeed. So we have a new objective: **maximize users' engagement** while **minimizing the spread of extreme views and misinformation**. So we added two new objectives:

1. Filter out spam
2. Filter out NSFW content
3. Filter out misinformation
4. Rank posts by quality (let's pretend for now that we know how to measure quality)
5. Rank posts by engagement: how likely users will click on it

Multiple objective optimization (MOO)

Now our objective 4 and 5 might be conflicting with each other. For example, a post might be very engaging but of questionable quality -- where should that post go?

Let's take a step back to see what exactly each objective does. To rank posts by quality, we first need to predict posts' quality, which can be achieved by trying to minimize **quality_loss**: the difference between each post's predicted quality and its true quality.

[8] Facebook Employee Raises Powered by 'Really Dangerous' Algorithm That Favors Angry Posts (SFist, 2019)
[9] The Making of a YouTube Radical (NYT, 2019)

Similarly, to rank posts by engagement, we first need to predict the number of clicks each post will get, which can be achieved by trying to minimize **engagement_loss**: the difference between each post's predicted clicks and the actual number of clicks it gets.

One approach is to combine these two losses into one loss and train one model to minimize that loss.

$$\textbf{loss} = \alpha\ \textbf{quality\_loss} + \beta\ \textbf{engagement\_loss}$$

You can tune tune α and β to meet your need. To learn how to choose α and β, you can check out Pareto optimization, *an area of multiple criteria decision making that is concerned with mathematical optimization problems involving more than one objective function to be optimized simultaneously*. Also check out this great paper on applying Pareto optimization for ML because, the authors claimed: "machine learning is inherently a multiobjective task[10]".

A problem with this approach is that each time you tune α and β, you'll have to retrain your model.

Another approach is to train two different models, each optimizing one loss. So you have:

- quality_model that minimizes quality_loss and outputs the predicted quality of each post
- engagement_model that minimizes engagement_loss and outputs the predicted number of clicks of each post

You can combine the outputs of these two models and rank posts by these combined values:

$$\alpha\ \textbf{quality\_model(post)} + \beta\ \textbf{engagement\_model(post)}$$

Now you can tweak α and β without retraining your models.

In general, when there are multiple objectives, it's might be a good idea to decouple them because:

- It's easier for training:
    - Optimizing for one objective is easier than optimizing for multiple objectives
- It's easier to tweak your system without retraining models
- It's easier for maintenance since different objectives might need different maintenance schedules. For example, spamming techniques evolve much faster than the way post quality is perceived, so spam filtering systems need updates at a much higher frequency than quality ranking systems.

Constraints

A project's constraints rule out impossible solutions.

[10] Pareto-Based Multiobjective Machine Learning: An Overview and Case Studies (Jin and Sendhoff, IEEE 2008)

- **Time**
- **Performance**
  - **Baselines**: What are you comparing the system to?
    - Example: existing solutions, simple solutions, human experts, competitors solutions, etc.
  - **Usefulness threshold**: how fast/good the solution needs to be for it to be useful?
    - Example: self-driving cars need human-level performance to be useful. A system that predicts what word a user will type next on their phone doesn't.
  - **Performance tradeoffs**: what's more important -- false negatives or false positives?
    - Example: covid screening must not have false negative (patients with covid shouldn't be classified as no covid). Fingerprint unlocking must not have false positive (unauthorized people shouldn't be given access)
  - **Interpretability:** Does the solution need to be interpretable? If yes, to whom?
  - **Confidence measurement**: Does the solution need confidence measurement, e.g. how confident the model is about a prediction?
    - If yes, what is the confidence threshold a prediction has to pass to be usable? What do you want to do with predictions below that threshold—discard it, loop in humans, or ask for more information from users?
- **Budget**
  - What is the maximum initial budget?
  - What needs to be accomplished to increase the budget?
- **Stakeholders**
  - Who, both internal and external, will be affected by the project?
  - Who needs to be informed about the project?
- **Privacy**
  - Privacy requirements for annotation, storage, third-party solutions, cloud services, regulations
    - Can data be shipped outside organizations for annotation?
    - Can the system be connected to the Internet?
    - How long can you keep users data?
- **Technical constraints**
  - Is there any tool or system that the solution must be compatible with?
    - Legacy infrastructure is a huge bottleneck for organizations to adopt ML.

Evaluation

- How to evaluate the system's performance, both during development and deployment?

    - If you need ground truth labels, how can they be generated or inferred from users' reactions?
- How to tie model performance to business goals and objectives?

When to use machine learning

Despite an incredible amount of excitement and hype generated by people both inside and outside the field, ML is not a magic tool that can solve all problems. Even for problems that ML can solve, ML solutions might not be the optimal solutions.

Before starting an ML project, you might want to ask whether ML is necessary[11] and whether the cost-benefit equation for ML makes sense.

To understand what problems ML can solve, let's take a step back and understand what ML is:

Machine learning is an approach to learn complex patterns from existing data and use these patterns to make predictions on unseen data.

We'll look at each of the underlined keyphrases in the definition to understand its implications to the problems ML can solve.

1. **Patterns: there are patterns to learn**
   It'd be foolish to build an ML system to predict the next outcome of a fair die or the next winning lottery ticket because there's no pattern in how these outcomes are generated[12].

   Similarly, it wouldn't make sense to build a system to predict the price of a stock if you believed the price is entirely random. However, since there are patterns in the stock market, companies have invested billions of dollars in building ML systems to learn those patterns.

2. **Learn: it's possible to form an objective function to guide the learning**
   Given a dataset, an ML algorithm might learn useful patterns or it might mistake noises for patterns. Objective functions guide ML algorithms in learning because ML algorithms update candidate models to minimize/maximize these functions.

   For supervised learning, an objective function can be defined by the differences between the ground truth labels and the predictions made by candidate models. The differences

[11] I didn't ask whether ML is sufficient because the answer is always no.
[12] Patterns are different from distributions. We know the distribution of the outcomes of a fair die, but there are no patterns in the way the outcomes are generated.

have to be mathematically quantified, such as mean squared error or negative log-likelihood.

3. **Complex: the patterns are complex**
   Consider a website like Airbnb with a lot of house listings, each listing comes with a zip code. If you want to sort listings into the states they are located in, you wouldn't need an ML system. Since the pattern is simple—each zip code corresponds to a known state—you can just use a lookup table.

   However, predicting the price of each listing requires much more complex patterns. The price of the house depends on the neighborhood, how big the lot is, how many bedrooms the house has, the year it was built, nearby schools, nearby listings, demands for housing in that area, risks of natural disasters, mortgage rate, the housing market, HAO fee, etc. With good training practices, ML algorithms can learn these patterns from data and predict the price of a new listing without you having to explicitly write out the rules.

   ML has been very successful with tasks with complex patterns such as object detection and speech recognition. Algorithmic complexity is different from complexity in human perception. Many tasks that are hard for humans to do can be easy to express in algorithms, e.g. raising a number of the power of 10. Vice versa, many tasks that are easy for humans can be hard to express in algorithms, e.g. deciding whether there's a cat in a picture.

4. **Existing data: data is available, or it's possible to collect data**
   Since ML learns from data, there must be data for it to learn from. It's amusing to think about building a model to predict how much tax a person should pay a year, but it's not possible unless you have access to tax and income data of a large population[13].

   Existing data can be public, proprietary, or synthesized. Or, you can follow a 'fake-it-til-you make it' approach: launching a product that serves predictions made by humans, instead of ML algorithms, with the hope of using the generated data to train ML algorithms.

5. **Unseen data: Unseen data shares patterns with the training data**
   The patterns your model learns from existing data are only useful if unseen data also share these patterns. A model to predict whether an app will get downloaded on Christmas 2020 won't perform very well if it's trained on data from 2008 when the most popular app on the App Store was Koi Pond. What's Koi Pond? Exactly.

[13] Looking at you, IRS.

In technical terms, it means your unseen data and training data should come from similar distributions. You might ask: "If the data is unseen, how do we know what distribution it comes from?" We don't, but we can make assumptions—e.g. we can assume that users' behaviors tomorrow won't be too different from users' behaviors today—and hope that our assumptions hold. If they don't, we'll find out soon enough.

6. **<u>Predictions</u>: it's a predictive problem**
   ML algorithms make predictions, so they can only solve problems that require predictions. ML can be especially appealing when you can benefit from a large quantity of cheap but approximate predictions. Semantically, "predict" means estimating something in the future—what would the weather be like tomorrow? What would win the Super Bowl this year? What movie would a user want to watch next?

   As predictive machines (e.g. ML models) are becoming more effective, more and more problems are being reframed as predictive problems. Whatever question you might have, you can always frame it as: "What would the answer to this question be?"

   Computing problems are one class of problems that have been very successfully reframed as predictive. Instead of computing the exact outcome of a process, which might be even more computationally costly and time-consuming than ML, you can frame the problem as: "What would the outcome of this process look like?" and approximate it using an ML algorithm. The output will be an approximation of the exact output, but often, it's good enough. You can see a lot of it in graphic renderings, such as image denoising[14], screen-space shading[15].

Due to the way machines learn, ML solutions will especially shine if your problem has these additional following characteristics.

7. **It's repetitive**
   Humans are great at few-shot learning: you can show kids a few pictures of cats and most of them will recognize a cat the next time they see one. Despite exciting progress in few-shots learning research, most ML algorithms still require many examples to learn a pattern. When a task is repetitive, each pattern is repeated multiple times, which makes it easier for machines to learn it.

8. **It's at scale**

[14] Kernel-predicting convolutional networks for denoising Monte Carlo renderings (Bako et al., ACM Transactions on Graphics 2017)
[15] Deep Shading: Convolutional Neural Networks for Screen-Space Shading (Nalbach et al., 2016)

ML solutions often require non-trivial upfront investment on data, compute, infrastructure, and talent, so it'd make sense if we can use these solutions a lot.

"At scale" means different things for different tasks, but it might mean making a lot of predictions. Examples include sorting through millions of mails a year or predicting which departments thousands of support tickets should be sent to a day.

A problem might appear to be a singular prediction but it's actually a series of predictions. For example, a model that predicts who will win a US presidential election seems like it only makes one prediction every four years, but it might actually be making a prediction every hour or even less because that prediction has to be updated to new information over time.

Having a problem at scale also means that there's a lot of data for you to collect, which is useful for training ML models.

9. **The patterns are constantly changing**
Cultures change. Tastes change. Technologies change. What's trendy today might be old news tomorrow. Consider the task of email spam classification. Today, an indication of a spam email is a Nigerian prince but tomorrow it might be a distraught Vietnamese writer.

If your problem involves one or more constantly changing patterns, solutions that don't allow you to learn from changing data might get you stuck in the past.

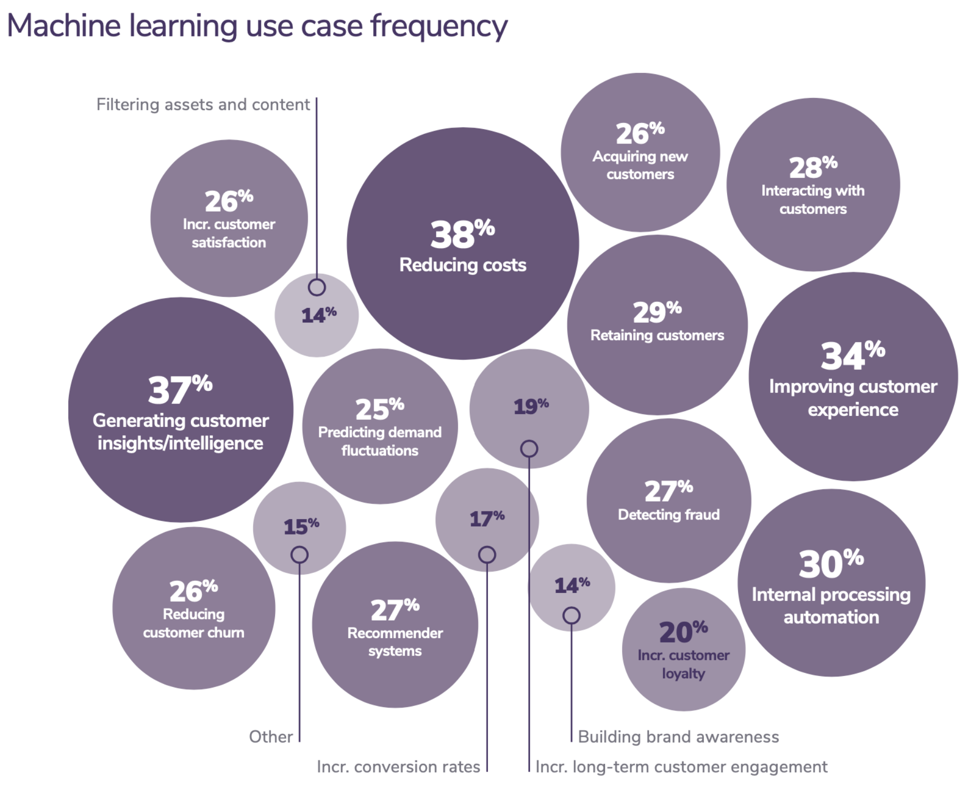
Example use cases

ML has found increasing usage in both enterprise and end-user applications. Since the mid-2020s, there has been an explosion of applications that leverage ML to deliver superior or previously impossible services to the end-users. Below are some of the notable examples.

- **Authentication** with your face or fingerprints
- **Machine translation**
- **Writing**: autocorrection, autocompletion, predictive typing
- **Photos**: auto-enhancements and filters
    - E.g. Google Photos, Instagram, Snapchat
- **Personal assistant**: automatically scheduling, booking flight tickets, placing orders, question answering
    - E.g. Siri, Google Assistant, Alexa
- **Home security**: detecting when pets leave house or intrusion when no one is home
- **Elderly care**: at-home health monitoring, fall detection
- **Health care**: skin cancer detection, diabetes diagnosis, drug-drug interaction predictions, drug discovery

- **General amusements**: finding your celebrity doppelganger, converting your face into a cartoon character, changing your voice

Even though the market for ML applications for end-users is booming, the majority of ML use cases are still in the enterprise world. According to Algorithmia's 2020 state of enterprise machine learning survey, ML applications in enterprises are diverse, serving both internal use cases (reducing costs, generating customer insights and intelligence, internal processing automation) and external use cases (improving customer experience, retaining customers, interacting with customers).[16]



2020 state of enterprise machine learning. Algorithmia 2020.

Below are some of the problems that have found a wide adoption of ML in enterprises.

- **Search engine**: both as a standalone search engine like Google or as part of another service such as the search functionality for app stores, Amazon, Netflix, etc.
- **Recommendation systems**: recommending what content a user should consume next on a platform (e.g. movies you might like, related products). A good recommender system can lead to higher customer engagement, more sales, better customer satisfaction.
- **Demand forecasting**: to run a business, it's important to be able to forecast customer demand so that you can prepare a budget, stock inventory, allocate resources, and update

[16] 2020 state of enterprise machine learning (Algorithmia, 2020)

pricing strategy. For example, if you run a grocery store, you want to stock enough so that customers find what they're looking for, but you don't want to stock too much that your groceries go bad.

- **Pricing optimization**: deciding how much to charge for your product or service is probably one of the hardest business decisions, why not let ML do it for you? Price optimization is the process of estimating a price at a certain time period to maximize a defined objective function, e.g. the company's margin or revenue. ML-based pricing optimization is most suitable for cases with a large number of transactions where demand fluctuates and consumers are willing to pay a dynamic price e.g. Internet ads, flight tickets, accommodation bookings, ride-sharing, events.
- **Customer acquisition**: acquiring a new user is expensive. As of 2019, the average cost for an app to acquire a user who'll make an in-app purchase is $86.61[17]. The acquisition cost for Lyft is estimated at $158/rider[18]. This cost is so much higher for enterprise customers. Customer acquisition cost is hailed by investors as a startup killer[19]. Reducing customer acquisition costs by a small amount can result in a large increase in profit. This can be done through better identifying potential customers, showing better-targeted ads, giving discounts at the right time, etc.—all of which are suitable tasks for ML.
- **Churn prediction**: after you've spent so much money acquiring a customer, it'd be a shame if they leave. Churn prediction is predicting when a specific customer is about to stop using your products or services so that you can take appropriate actions to win them back. Churn prediction can be used not only for customers but also for employees.
- **Support ticket classification**: when a customer opens a support ticket or sends an email, it usually needs to first be processed then passed around to different departments until it arrives at the inbox of someone who can address it. This process can be automated with an ML system to analyze the ticket content and predict where it should go, which can shorten the response time and improve customer satisfaction. It can also be used to classify internal IT tickets.
- **Fraud detection**: this is among the oldest applications of ML in the industry. If your product or service involves transactions of any value, it'll be susceptible to fraud. By leveraging ML solutions for anomaly detection, you can have systems that learn from historical fraud transactions and predict whether a future transaction is fraudulent.
- **Brand monitoring**: the brand is a valuable asset of a business[20]. It's important to monitor how the public and how your customers perceive your brand. You might want to know when/where/how it's mentioned, both explicitly (e.g. when someone mentions "Google") or implicitly (e.g. when someone says "the search giant") as well as the sentiment

[17] Average mobile app user acquisition costs worldwide from September 2018 to August 2019, by user action and operating system (Statista, 2019)
[18] Valuing Lyft Requires A Deep Look Into Unit Economics (Forbes, 2019)
[19] Startup Killer: the Cost of Customer Acquisition (David Skok, 2018)
[20] Apple, Google, Microsoft, Amazon each has a brand estimated to be worth in the order of hundreds of millions dollars (Forbes, 2020)

associated with it. If there's suddenly a surge of negative sentiment in your brand mentions, you might want to do something about it as soon as possible. Sentiment analysis is a typical ML task.

When not to use machine learning

The list of use cases can go on and on, and it'll grow even longer as ML adoption matures in the industry. Even though ML can solve a subset of problems very well, it can't solve and/or shouldn't be used for a lot of problems. ML shouldn't be used if under any of the following conditions.

1. It's unethical.
2. Simpler solutions do the trick.
3. It's impossible to get the right data.
4. One single prediction error can cause devastating consequences.
5. Every single decision the system makes must be explainable.
6. It's not cost-effective.

However, even if ML can't solve your problem, it might be possible to break your problem into smaller components and ML can solve some of them. For example, if you can't build a chatbot to answer all your customers' queries, it might be possible to build an ML model to predict whether a query matches one of the frequently asked questions. If yes, automatically direct the customer to the answer. If not, direct them to customer service.

I'd also want to caution against dismissing a new technology because it's not as cost-effective as older technologies at the moment. Most technological advances are incremental. A technology might not be efficient now, but it might be in the future. If you wait for the technology to prove its worth to the rest of the industry before jumping on, you might be years or decades behind your competitors.

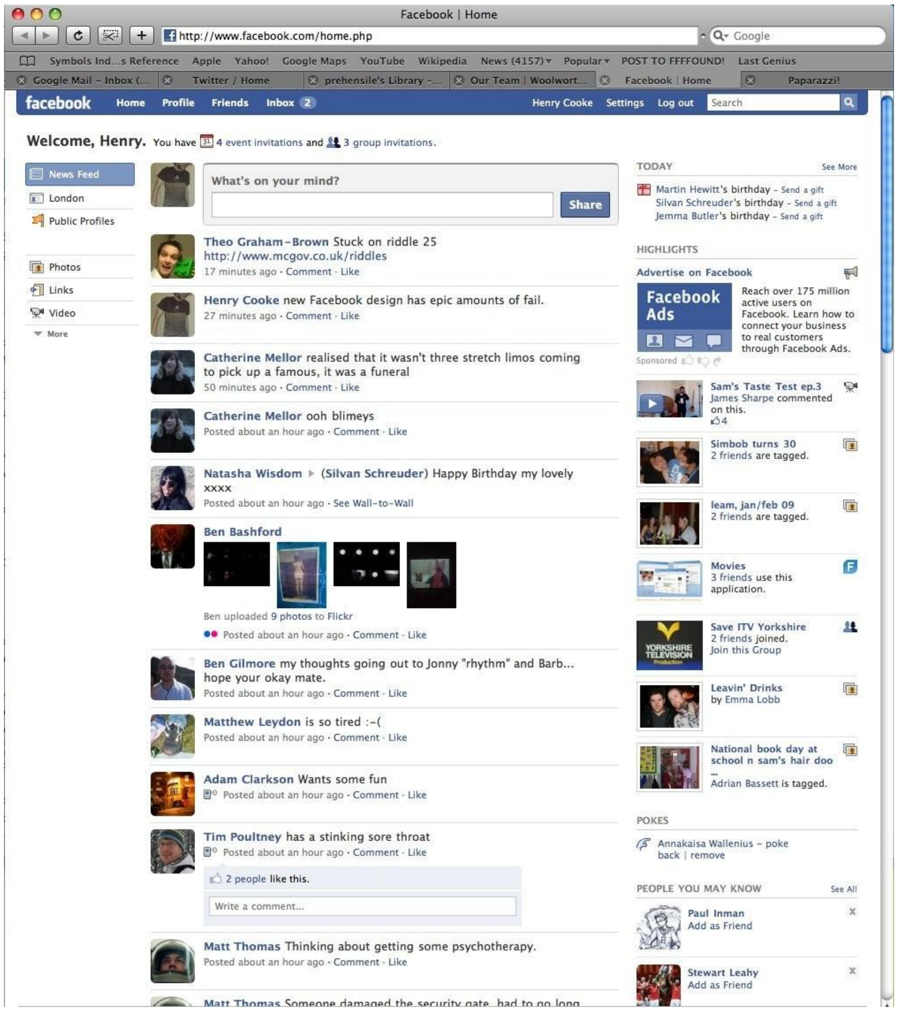
Four phases of machine learning adoption

Once you've decided to explore ML, your strategy depends on which phase of ML adoption you are in. There are four phases of adopting ML, with solutions from each phase can be used as baselines to evaluate the solutions from the next phase.

Phase 1: Before machine learning

If this is your first time trying to make this type of prediction from this type of data, start with non-ML solutions. Your first stab at the problem can be the simplest heuristics. For example, to predict what letter users are going to type next in English, you can show the top three most common English letters, "e", "t", and "a", which is correct 30% of the time.

Facebook newsfeed was introduced in 2006 without any intelligent algorithms—posts were shown in a chronological order. It wasn't until 2011 that Facebook started displaying news updates you were most interested in at the top of the feed[21].


Facebook | Home
http://www.facebook.com/home.php
Google
Symbols Ind...s Reference Apple Yahoo! Google Maps YouTube Wikipedia News (4157) Popular POST TO FFFFOUND! Last Genius
Google Mail – Inbox (... Twitter / Home prehensile's Library –... Our Team | Woolwort... Facebook | Home Paparazzi!
facebook Home Profile Friends Inbox 2 Henry Cooke Settings Log out Search
News Feed
London
Public Profiles
Photos
Links
Video
More
What's on your mind?
Share

Welcome, Henry. You have 4 event invitations and 3 group invitations.

Theo Graham-Brown Stuck on riddle 25
http://www.mcgov.co.uk/riddles
17 minutes ago · Comment · Like

Henry Cooke new Facebook design has epic amounts of fail.
27 minutes ago · Comment · Like

Catherine Mellor realised that it wasn't three stretch limos coming to pick up a famous, it was a funeral
50 minutes ago · Comment · Like

Catherine Mellor ooh blimeys
Posted about an hour ago · Comment · Like

Natasha Wisdom ▸ (Silvan Schreuder) Happy Birthday my lovely xxxx
Posted about an hour ago · See Wall-to-Wall

Ben Bashford
Ben uploaded 9 photos to Flickr
Posted about an hour ago · Comment · Like

Ben Gilmore my thoughts going out to Jonny "rhythm" and Barb... hope your okay mate.
Posted about an hour ago · Comment · Like

Matthew Leydon is so tired :-(
Posted about an hour ago · Comment · Like

Adam Clarkson Wants some fun
Posted about an hour ago · Comment · Like

Tim Poultney has a stinking sore throat
Posted about an hour ago · Comment · Like
2 people like this.
Write a comment...

Matt Thomas Thinking about getting some psychotherapy.
Posted about an hour ago · Comment · Like

Matt Thomas Someone damaged the security gate, had to go long

TODAY See More
Martin Hewitt's birthday – Send a gift
Silvan Schreuder's birthday – Send a gift
Jemma Butler's birthday – Send a gift

HIGHLIGHTS
Advertise on Facebook
Facebook Ads
Reach over 175 million active users on Facebook. Learn how to connect your business to real customers through Facebook Ads.
Sponsored

Sam's Taste Test ep.3
James Sharpe commented on this.
4

Simbob turns 30
2 friends are tagged.

leam, jan/feb 09
2 friends are tagged.

Movies
3 friends use this application.

Save ITV Yorkshire
2 friends joined.
Join this Group

Leavin' Drinks
by Emma Lobb

National book day at school n sam's hair doo
...
Adrian Bassett is tagged.

POKES
Annakaisa Wallenius – poke back | remove

PEOPLE YOU MAY KNOW See All
Paul Inman
Add as Friend
Stewart Leahy
Add as Friend

Facebook newsfeed circa 2006

According to Martin Zinkevich in his magnificent *Rules of Machine Learning: Best Practices for ML Engineering*:

"If you think that machine learning will give you a 100% boost, then a heuristic will get you 50% of the way there.[22]"

You might find out that non-ML solutions work just fine and you don't need ML yet.

[21] The Evolution of Facebook News Feed (Samantha Murphy, Mashable 2013)
[22] Rules of Machine Learning: Best Practices for ML Engineering (Martin Zinkevich, Google 2019)

Phase 2: Simplest machine learning models

For your first ML model, you want to start with a simple algorithm, something that gives you visibility into its working to allow you to validate the usefulness of your problem framing and your data. Logistic regression XGBoost, K-nearest neighbors can be great for that.

They are also easier to implement and deploy which allows you to quickly build out a framework from data management to development to deployment that you can test and trust.

Phase 3: Optimizing simple models

Once you've had your ML framework in place, you can focus on optimizing the simple ML models with different objective functions, hyperparameter search, feature engineering, more data, ensembles.

This phase will allow you to answer questions such as how quickly your model decays in production and update your infrastructure accordingly.

Phase 4: Complex systems

Once you've reached the limit of your simple models and your use case demands significant model improvement, experiment with more complex models.

5. Case studies

To learn to design ML systems, it's helpful to read case studies to see how actual teams deal with different deployment requirements and constraints. Many companies—Airbnb, Lyft, Uber, and Netflix, to name a few—run excellent tech blogs where they share their experience using ML to improve their products and/or processes.

1. Using Machine Learning to Predict Value of Homes On Airbnb (Robert Chang, Airbnb Engineering & Data Science, 2017)
In this detailed and well-written blog post, Chang described how Airbnb used machine learning to predict an important business metric: the value of homes on Airbnb. It walks you through the entire workflow: feature engineering, model selection, prototyping, moving prototypes to production. It's completed with lessons learned, tools used, and code snippets too.

2. Using Machine Learning to Improve Streaming Quality at Netflix (Chaitanya Ekanadham, Netflix Technology Blog, 2018)
As of 2018, Netflix streams to over 117M members worldwide, half of those living outside the US. This blog post describes some of their technical challenges and how they use machine learning to overcome these challenges, including to predict the network quality, detect device anomaly, and allocate resources for predictive caching.

3. 150 Successful Machine Learning Models: 6 Lessons Learned at Booking.com (Bernardi et al., KDD, 2019)
As of 2019, Booking.com has around 150 machine learning models in production. These models solve a wide range of prediction problems (e.g. predicting users' travel preferences and how many people they travel with) and optimization problems (e.g.optimizing the background images and reviews to show for each user). Adrian Colyer gave a good summary of the six lessons learned here:

1. Machine learned models deliver strong business value.
2. Model performance is not the same as business performance.
3. Be clear about the problem you're trying to solve.
4. Prediction serving latency matters.
5. Get early feedback on model quality.
6. Test the business impact of your models using randomized controlled trials.

4. How we grew from 0 to 4 million women on our fashion app, with a vertical machine learning approach (Gabriel Aldamiz, HackerNoon, 2018)
To offer automated outfit advice, Chicisimo tried to qualify people's fashion taste using machine learning. Due to the ambiguous nature of the task, the biggest challenges are framing the problem and collecting the data for it, both challenges are addressed by the article. It also covers the problem that every consumer app struggles with: user retention.

5. Machine Learning-Powered Search Ranking of Airbnb Experiences (Mihajlo Grbovic, Airbnb Engineering & Data Science, 2019)
This article walks you step by step through a canonical example of the ranking and recommendation problem. The four main steps are system design, personalization, online scoring, and business aspect. The article explains which features to use, how to collect data and label it, why they chose Gradient Boosted Decision Tree, which testing metrics to use, what heuristics to take into account while ranking results, how to do A/B testing during deployment. Another wonderful thing about this post is that it also covers personalization to rank results differently for different users.

6. From shallow to deep learning in fraud (Hao Yi Ong, Lyft Engineering, 2018)
Fraud detection is one of the earliest use cases of machine learning in the industry. This article explores the evolution of fraud detection algorithms used at Lyft. At first, an algorithm as simple as logistic regression with engineered features was enough to catch most fraud cases. Its simplicity allowed the team to understand the importance of different features. Later, when fraud techniques have become too sophisticated, more complex models are required. This article explores the tradeoff between complexity and interpretability, performance and ease of deployment.

7. Space, Time and Groceries (Jeremy Stanley, Tech at Instacart, 2017)
Instacart uses machine learning to solve the task of path optimization: how to most efficiently assign tasks for multiple shoppers and find the optimal paths for them. The article explains the entire process of system design, from framing the problem, collecting data, algorithm and metric selection, topped with a tutorial for beautiful visualization.

8. Creating a Modern OCR Pipeline Using Computer Vision and Deep Learning (Brad Neuberg, Dropbox Engineering, 2017)
An application as simple as a document scanner has two distinct components: optical character recognition and word detector. Each requires its own production pipeline, and the end-to-end system requires additional steps for training and tuning. This article also goes into detail the team's effort to collect data, which includes building their own data annotation platform.

9. Scaling Machine Learning at Uber with Michelangelo (Jeremy Hermann and Mike Del Balso, Uber Engineering, 2019)
Uber uses extensive machine learning in their production, and this article gives an impressive overview of their end-to-end workflow, where machine learning is being applied at Uber, and how their teams are organized.

10. Spotify's Discover Weekly: How machine learning finds your new music (Umesh .A Bhat, 2017)
To create Discover Weekly, there are three main types of recommendation models that Spotify employs:

- **Collaborative Filtering** models (i.e. the ones that Last.fm originally used), which work by analyzing your behavior and others' behavior.
- **Natural Language Processing** (NLP) models, which work by analyzing text.
- **Audio** models, which work by analyzing the raw audio tracks themselves.

==

To cite this lecture note, please use:

```
@booklet{cs329s_lectures,
  title  = {CS 329S: Machine learning systems design},
  author = {Chip Huyen},
  url    = {https://cs329s.stanford.edu},
  year   = {2021},
  note   = {Lecture notes}
}
```