

# Introduction to Machine Learning in Production

In the first course of Machine Learning Engineering for Production Specialization, you will identify the various components and design an ML production system end-to-end: project scoping, data needs, modelling strategies, and deployment constraints and requirements; and learn how to establish a model baseline, address concept drift, and prototype the process for developing, deploying, and continuously improving a productionized ML application.

Understanding machine learning and deep learning concepts is essential, but if you're looking to build an effective AI career, you need production engineering capabilities as well. Machine learning engineering for production combines the foundational concepts of machine learning with the functional expertise of modern software development and engineering roles to help you develop production-ready skills. Week 1: Overview of the ML Lifecycle and Deployment Week 2: Selecting and Training a Model Week 3: Data Definition and Baseline

## Week 1: Overview of the ML Lifecycle and Deployment

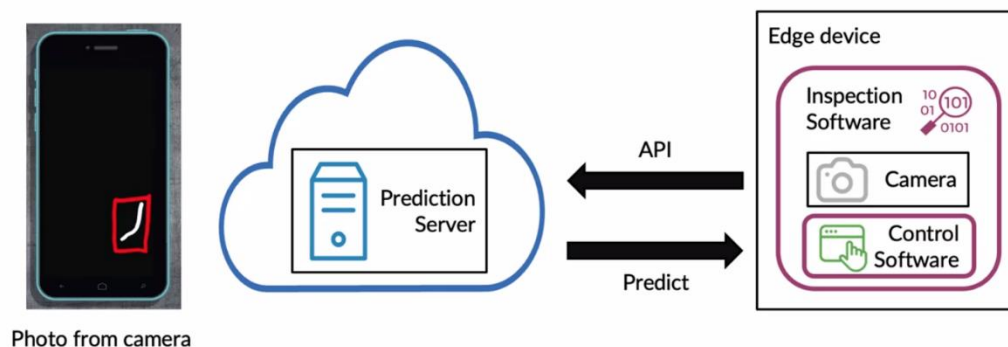
### Contents

<b>Week 1: Overview of the ML Lifecycle and Deployment</b>	<b>1</b>
Introduction	2
Steps of an ML Project	4
Case Study – Speech Recognition	5
Deployment – Key Challenges	8
Deployment Patterns	12
Monitoring	15
Pipeline Monitoring	19
Labs	21
Reading References	21

## Introduction

- There are machine learning challenges to production as well that aren't just about the process of developing a model in a Jupyter notebook.
- You can look at production machine learning as both machine learning itself and the knowledge and skills required in modern software development.
- If you're working on a machine learning team and industry, you really need expertise in both machine learning and software to be successful. This is because your team will not just be producing a single result. You'll be developing a product or service that will operate continuously as part of a mission critical part of your company's work
- Oftentimes the most challenging aspects of building machine learning systems turn out to be things that you least expect, like deployment.
- It's all very well being able to build a model, but getting that into people's hands and seeing how they use it can be very eye-opening
- Machine learning models are great, but unless you know how to put them into production, it's hard to get them to create the maximum amount of possible value.

## Deployment example



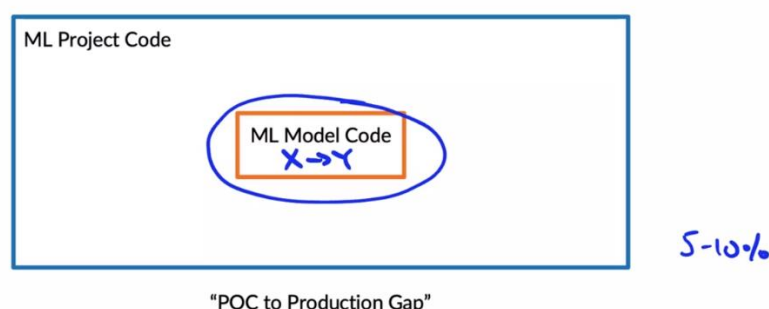
- This is actually commonly done in factories is called automated visual defect inspection.
- What the inspection software does is it will control camera that will take a picture of the smartphone as it rolls off the manufacturing line.
- And it then has to make an API call to pass this picture to a **prediction server**. And the job of the prediction server is to accept these API calls, receive an image, make a decision as to whether or not this phone is defective and return this prediction.
- And then the inspection control software can make the appropriate control decision whether to let it still move on in the manufacturing line.
- Or whether to shove it to a side, because it was defective and not acceptable.
- You have to take this machine learning model, put it in a production server, setup API interfaces and write all of the rest of the software in order to deploy this learning algorithm into production.
- This prediction server is sometimes in the cloud and sometimes the prediction server is actually at the edge as well.
- In fact in manufacturing we use **edge** deployments a lot, because you can't have your factory go down every time your internet access goes down.

## Visual inspection example



- In this scenario, there's a good phone on the left, and the one in the middle has a big scratch across it, and you've trained your learning algorithm to recognize that things like this on the left are okay, while drawing bounding boxes around scratches or other defects that the model finds (for the image in the middle)
- When you deploy it in the factory, you may find that the real life production deployment gives you back images which are much darker (image on the right) because the lighting conditions in the factory have changed for some reason compared to the time when the training set was collected.
- This problem is sometimes called **concept drift** or **data drift**.
- This is one example of the many practical problems that we, as machine learning engineers should step up to solve if we want to make sure that we don't just do well on the holdout test set, but that our systems actually create value in a practical production deployment environment.
- Sometimes I'll see many projects with success in development environment, still takes maybe another six months of work for practical deployment.
- This is just one of many of the practical things that a machine learning team has to watch out for and handle in order to actually deploy these systems.

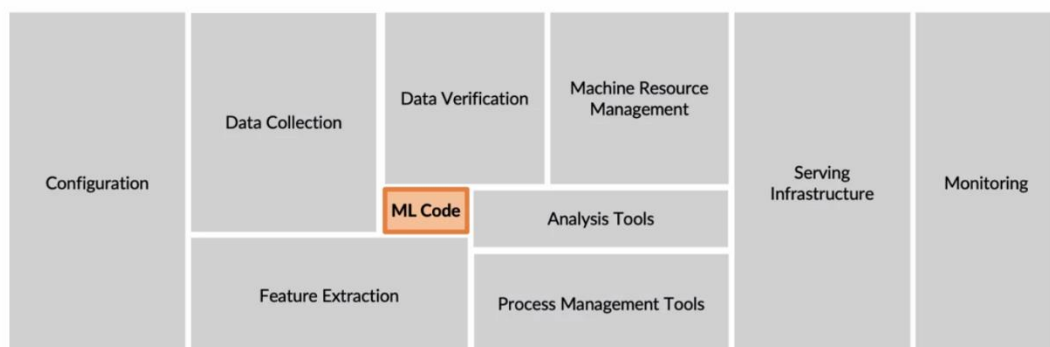
## ML in production



- A second challenge of deploying machine learning models and production is that it takes a lot more than machine learning code.
- Over the last decade there's been a lot of attention on machine learning models (e.g. neural networks or other algorithms that learns a function mapping from some input to some output), and there has been amazing progress in machine learning models.

- But it turns out that if you look at a machine learning system in production, this little orange rectangle represents the machine learning code, the machine learning model code, and the entire blue rectangle represents all the codes you need for the entire machine learning project.
- For many machine learning projects, maybe only 5-10% is actually machine learning code.
- One of the reasons why when you have a proof of concept model working in Jupyter notebook, it can still take a lot of work to go from that initial proof of concept to the production deployment. Sometimes people refer to it as the proof of concept (POC) to production gap.
- And a lot of that gap is sometimes just the sheer amount of work it takes to write all of this code out here beyond the initial machine learning model code.

## The requirements surrounding ML infrastructure



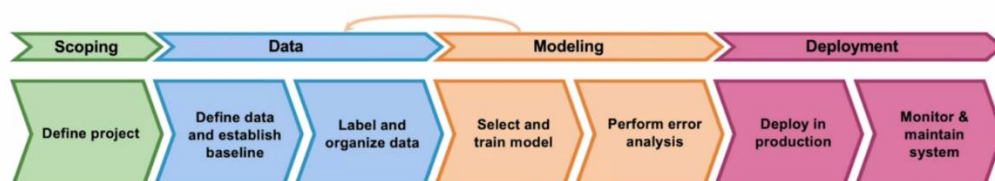
[D. Sculley et. al. NIPS 2015: Hidden Technical Debt in Machine Learning Systems] 

- Beyond the machine learning codes there are also many components, especially components for managing the data, such as data collection, data verification, feature extraction.
- And after you are serving it, we also need to consider how to monitor and analyse the system. There are often many other components that need to be built to enable a working production deployment.
- So in this course you learn what are all of these other pieces of software needed for a valuable production deployment

## Steps of an ML Project

- When I'm building a machine learning system, thinking through the Machine Learning project lifecycle is an effective way for me to plan out all the steps that I need to work on.
- When you are working on Machine Learning system, I think you'll find too that this framework allows you to plan out all the important things you need to do in order to get the system to work, and also to minimize surprises.

## The ML project lifecycle

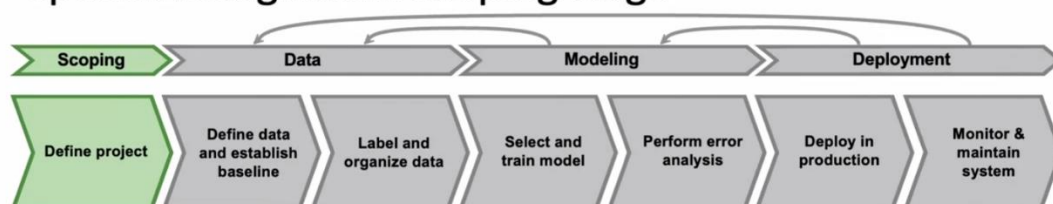


- First is **scoping**, in which you have to define the project or decide what to work on.
- What exactly do you want to apply Machine Learning to, and what is X and what is Y.
- After choosing the project, you have to acquire the data you need for your algorithm. This includes defining the data and establishing a baseline, and then labelling and organizing the data.
- After you have your data, you then have to train the model. During the model phase, you have to select and train the model and also perform error analysis. You might know that Machine Learning is often a highly iterative task. During the process of error analysis, you may go back and update the model, or you may also go back to the earlier phase and decide you need to collect more data as well.
- As part of error analysis before taking a system to deployments, I'll often also carry out a final check/audit, to make sure that the system's performance is good enough and that it's sufficiently reliable for the application.
- Sometimes, an engineer thinks that when you deploy a system, you're done.
- I now tell most people, **when you deploy a system for the first time, you are only about halfway to the finish line**, because it's often only after you turn on live traffic that you then learn the second half of the important lessons needed in order to get the system to perform well.
- To carry out the deployment step, you have to deploy it in production, write the software needed to put into production, then also monitor the system, track the data that continues to come in, and maintain the system.
- For example, if the data distribution changes, you may need to update the model.
- After the initial deployment, maintenance will often mean going back to perform more error analysis and maybe retrain the model, or it might mean taking the data you get back.
- Now that the system is deployed and is running on live data, feeding that output back into your dataset to then potentially update your data, retrain the model, and so on until you can put an updated model into deployment.
- Feel free to take a screenshot of this image and use it with your friends or by yourself to plan out your Machine Learning project as well.

## Case Study – Speech Recognition

- Let's use the machine learning project life cycle to set through a speech recognition example so you can understand all the steps needed to actually build and deploy such a system

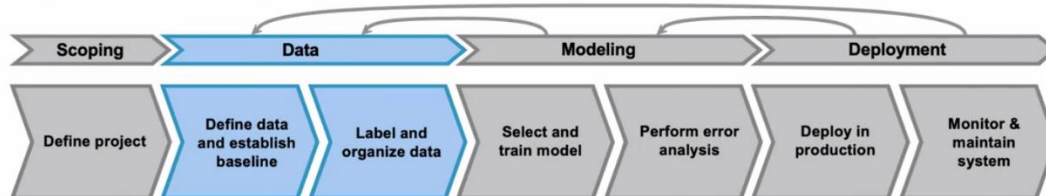
### Speech recognition: Scoping stage



- Decide to work on speech recognition for voice search.
- Decide on key metrics:
  - Accuracy, latency, throughput
- For scoping, we have to first define the project and just make a decision to work on speech recognition, say for voice search as part of defining the project.
- That also encourage you to try to estimate the key metrics. This will be very problem dependent. Almost every application will have his own unique set of goals and metrics.

- But the case of speech recognition, some things I cared about where how accurate is the speech system, what is the latency, how long does the system take to transcribe speech, what is the throughput, how many queries per second we handle.
- And then if possible, you might also try to estimate the resources needed. So how much time, how much compute how much budget as well as timeline. How long will it take to carry out this project?

## Speech recognition: Data stage



### Define data

- Is the data labeled consistently?
- How much silence before/after each clip? 100ms 300ms 500ms
- How to perform volume normalization?



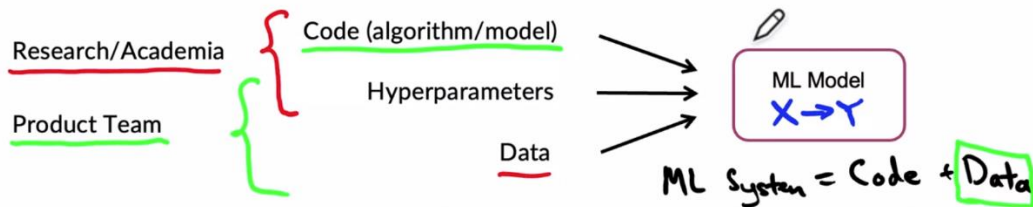
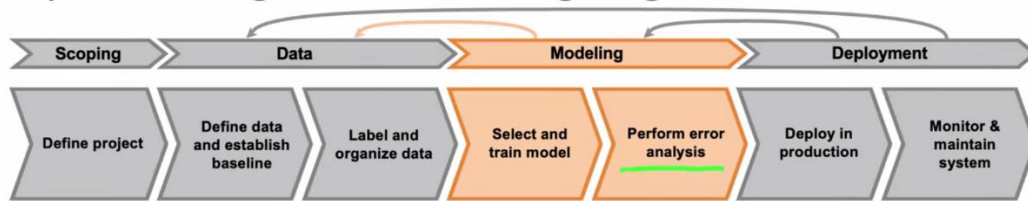
"Um, today's weather"

"Um... today's weather"

"Today's weather"

- The next step is the data stage where you have to define the data and establish a baseline and also label and organize the data. What's hard about this?
- One of the challenges of practical speech recognition systems is labelling the data consistently
- Here's an audio clip of a fairly typical recording you might get if you're working on speech recognition for voice search. And the question is given this audio clip that you just heard, how would you transcribe it? 1) "Um, today's weather", 2) "Um... today's weather" 3) "Today's weather".
- It turns out that any of these three ways of transcribing the audio is just fine and reasonable.
- I would probably prefer either the first or the second, not the third. But what hurts your learning algorithm's performance is if one third of the transcription used the first, one third used the second transcription, and one third using the third way of transcribing.
- Because then your data is inconsistent and confusing for the learning algorithm.
- How is the learning algorithm supposed to guess which one of these specific transcription to use for an audio clip?
- We can perhaps just ask everyone to standardize on the first convention. This will have a significant impact on your learning algorithm's performance.
- Other examples of data definition questions for an audio clip like how much silence do you want before and after each clip after a speaker has stopped speaking.
- Do you want to include another 100 milliseconds of silence after that? Or 300 milliseconds or 500 milliseconds, half a second? Or how do you perform volume normalization?
- Some speakers speak loudly, some are less loud and then there's actually a tricky case of a single audio clip with some really loud volume and some really soft volume, all within the audio clip.
- So how do you perform volume normalization questions like all of these are **data definition questions**.
- If you are working on a production system then you don't have to keep the data set fixed.
- I often edit the training set or even at the test set if that's what's needed in order to improve the data quality to get a production system to work better.

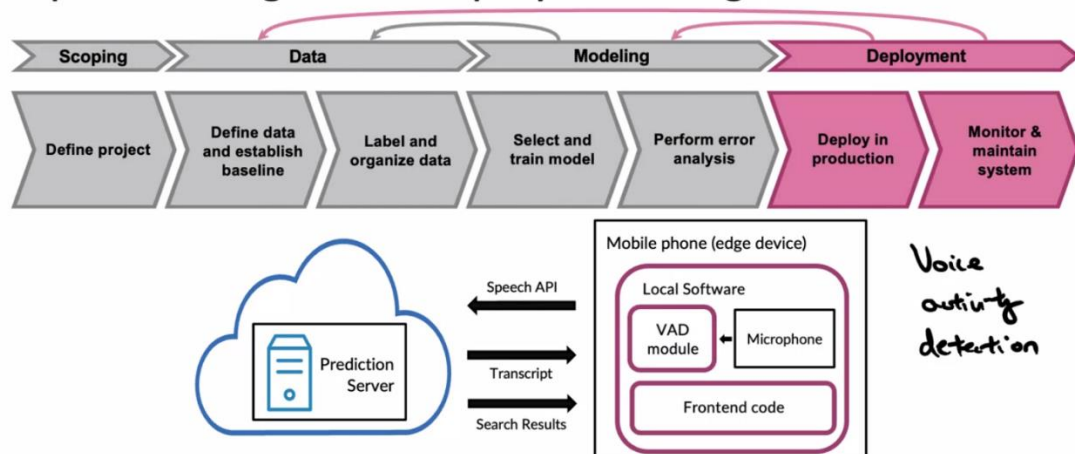
## Speech recognition: Modeling stage



- The three key inputs that go into training a machine learning model are the code that is the algorithm or the neural network model architecture that you might choose.
- You also have to pick hyperparameters and then there's the data and running the code with your hyper parameters on your data.
- I found that in a lot of research work or academic work you tend to hold the data fixed and vary the code and may be vary the hyperparameters in order to try to get good performance.
- In contrast, if your main goal is to just build and deploy a working machine learning system, I found that it can be even more effective to **hold the code fixed** and to instead focus on optimizing the data and maybe the hyper parameters in order to get a high performing model,
- A machine learning system includes both codes and data, and also hyperparameters which is maybe a bit easier to optimize than the code or data.
- And I found that rather than taking a model centric view of trying to optimize the code to your fixed data set for many problems, you can use an open source implementation of something you download of GitHub and instead just **focus on optimizing the data**.
- Error analysis can tell you where your model falls short, and how to systematically improve your data (and perhaps the code too)
- If the error analysis can tell you how to systematically improve the data, it can be a very efficient way for you to get to a high accuracy model.
- Part of the trick is you don't want to just feel like you need to collect more data all the time because we can always use more data.
- Rather than just trying to collect more and more and more data, which is helpful but can be **expensive**, if the error analysis can help you be more targeted in exactly what data to collect, it can help you be much more efficient in building an accurate model.



## Speech recognition: Deployment stage



- Finally, when you have trained the model and when error analysis seems to suggest is working well enough, you're then ready to go into deployment
- This is how you might deploy a speech system. You have a mobile phone. This would be an edge device with software running locally on your phone. That software taps into the microphone to record what someone is saying. For voice search and in a typical implementation of speech recognition, you would use a VAD module. VAD stands for a voice activity detection
- The job of the VAD allows the smartphone to select out just the audio that contains hopefully someone speaking so that you can send only that audio clip to your prediction server.
- And in this case maybe the prediction server lives into cloud. This would be a common deployment pattern. The prediction server then returns both the transcript to the user so you can see what the system thinks you said. And it also returns to search results.
- If you're doing voice search and the transcript and search results are then displayed in the frontend code running on your mobile phone.
- So implementing this type of system would be the work needed to deploy a speech model in production even after it's running though you still have to monitor and maintain the system.
- So here's something that happened to me once my team had built a speech recognition system and it was trained mainly on adult voices. We pushed into production, random production and we found that over time more and more young individuals, kind of teenagers, you know, sometimes even younger seem to be using our speech recognition system and the voices are very young individuals just sound different. And so my speech systems performance started to degrade. We just were not that good at recognizing speech as spoken by younger voices. And so we had to go back and find a way to collect more data are the things in order to fix it.
- So one of the key challenges when it comes to deployment is concept drift or data drift, which is what happens when the data distribution changes, such as there are more young voices being fed to the speech recognition system.
- Knowing how to put in place appropriate monitors to spot such problems and then also how to fix them in a timely way is a key skill needed to make sure your production deployment creates a value.

## Deployment – Key Challenges

- There are two major categories of challenges in deploying a machine learning model.
- First, are the machine learning or the statistical issues, and second, are the software engine issues.



# Concept drift and Data drift




## Speech recognition example

Training set:  $x \rightarrow y$

- Purchased data, historical user data with transcripts

Test set:

- Data from a few months ago

Gradual change  
Sudden shock 

How has the data changed?

- One of the challenges of a lot of deployments is, concept drift and, data drift. Loosely, this means what if your data changes after your system has already been deployed
- I trained a few speech recognition systems, and when I built speech systems, quite often I would have some purchased data. This would be some purchased or licensed data, which includes both the input  $x$ , the audio, as well as the transcript  $y$  that is the speech system output.
- I would collect a dev set or hold out validation set, as well as test set, comprising data from just the **last few months**. You can test it on fairly recent data to make sure your system works, even on relatively recent data.
- After you push the system to deployment, the question is, will the data change or after you've run it for a few weeks or a few months, has the data changed yet again?
- The data could have changed, such as the language changes or maybe people are using a brand new model of smartphone which has a different microphone, so the audio sounds different. This causes the performance of a speech recognition system to degrade.
- It's important for you to recognize how the data has changed, and if you need to update your learning algorithm as a result.
- When data changes, sometimes it is a **gradual change**, such as the English language which does change, but changes very slowly with new vocabulary introduced at a relatively slow rate.
- Sometimes data changes very suddenly where there's a **sudden shock** to a system.
- For example, when COVID-19 the pandemic hit, a lot of credit card fraud systems started to not work because the purchase patterns of individuals suddenly changed.
- Many people that did relatively little online shopping suddenly started to use much more online shopping. The way that people were using credit cards changed very suddenly, and this actually tripped up a lot of anti-fraud systems.
- This very sudden shift to the data distribution meant that many machine learning teams were scrambling a little bit at the start of COVID to collect new data and retrain systems in order to make them adapt to this very new data distribution.
- Another example of **Concept drift**, let's say that  $x$  is the size of a house, and  $y$  is the price of a house, because you're trying to estimate housing prices. If because of inflation or changes in the market, houses may become more expensive over time. The same sized house, will end up with a higher price.
- That would be **Concept drift**. Maybe the size of houses haven't changed, but the price of a given house changes.
- Whereas **Data drift** would be if, say, people start building larger houses, or start building smaller houses and thus the input distribution of the sizes of houses actually changes over time.
- When you deploy a machine learning system, one of the most important tasks, will often be to make sure you can detect and manage any changes.
- This includes both Concept drift, which is when the definition of what is  $y$  given  $x$ , changes.
- As well as **Data drift**, which is if the distribution of  $x$  changes, even if the mapping from  $x$  or  $y$  does not change.

# Software engineering issues

## Checklist of questions

- Realtime or Batch
- Cloud vs. Edge/Browser
- Compute resources (CPU/GPU/memory)
- Latency, throughput (QPS)
- Logging
- Security and privacy



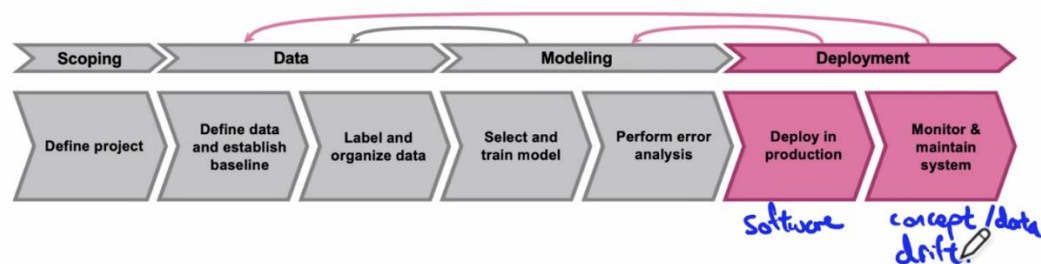
500ms, 1000 QPS



- Another set of issues are software engineering issues
- Let's say you are implementing a prediction service whose job it is to take queries X and output prediction Y.
- Here's a checklist of questions (above) that might help you with making the appropriate decisions for managing the software engineering issues.
- One decision you have to make for your application is: do you need Real time predictions or Batch predictions?
- For example, if you are building a speech recognition system, where the user speaks and you need to get a response back, in half a second, then clearly you need real time predictions.
- In contrast, I have also built systems for hospitals that take patient records. Take electronic health records: We run an overnight (once a night) batch process to see if there's something associated with the patients
- Whether you need to write real time software, where they can respond within hundreds of milliseconds or whether you can write software that just does a lot of computation overnight, it will affect how you implement your software.
- Second question you need to ask is, does your prediction service run in the cloud or does it run at the edge or maybe even in a Web browser?
- Today there are many speech recognition systems that run in the cloud, because having the compute resources of the cloud allows for more accurate speech recognition.
- There are also some speech systems, for example, a lot of speech systems within cars that actually run at the edge. There are also some mobile speech recognition systems that work, even if your Wi-Fi is turned off.
- When I am deploying visual inspection systems in factories, I pretty much almost always run that at the edge as well. Because sometimes unavoidably, the Internet connection between the factory, and the rest of the Internet may go down. You just can't afford to shut down the factory
- With the rise of modern Web browsers, there are better tools for deploying learning algorithms right there within a Web browser as well.
- When building a prediction service, it's also useful to take into account how much computer resources you have.
- There have been quite a few times where I trained a neural network on a very powerful GPU, only to realize that I couldn't afford an equally powerful set of GPUs for deployments. I had to do something else to compress or reduce the model complexity.
- If you know how much CPU or GPU resources and maybe also how much memory resources you have for your prediction service, then that could help you choose the right software architecture.
- Depending on your application, especially if it is a real-time application, latency and throughputs measured in terms of QPS (Queries per second) will be other software engineering metrics you may need to hit.

- In speech recognition it is not uncommon to want to get an answer back to the user within 500 milliseconds. Of this 500 millisecond budget you may be able to allocate only say, 300 milliseconds to your speech recognition. That gives a latency requirement for your system.
- Throughput refers to how many queries per second you need to handle given the compute resources,
- For example, if you're building a system that needs to handle 1000 queries per second, it would be useful to make sure to check out your system so that you have enough computer resources, to hit the QPS requirement.
- Next is logging, when building your system it may be useful to log as much of the data as possible for analysis and review as well as to provide more data for retraining your learning algorithm in the future.
- Finally, security and privacy, I find it for different applications the required levels of security and privacy can be very different. For example, when I was working on electronic health records, patient records, clearly the requirements for security and privacy were very high because patient records are very highly sensitive information.
- If you save this checklist somewhere, going through this when you're designing your software might help you to make the appropriate software engine choices when implementing your prediction service.

## First deployment vs. maintenance



- To summarize, deploying a system requires two broad sets of tasks: there is writing the software to enable you to deploy the system in production. There is what you need to do to monitor the system performance and to continue to maintain it, especially in the face of concepts drift as well as data drift.
- One of the things you see when you're building machine learning systems is that the practices for the very first deployments will be quite different compared to when you are updating or maintaining a system that has already previously been deployed.
- I know that to some engineers that view deploying the machine learning model as getting to the finish line. Unfortunately, I think the first deployment means you may be only about halfway there, and the second half of your work is just starting only after your first deployment. This is because even after you've deployed, there's a lot of work to feed the data back and maybe to update the model, to keep on maintaining the model even in the face of changes to the data.

## Deployment Patterns

### Common deployment cases

1. New product/capability
2. Automate/assist with manual task
3. Replace previous ML system

Key ideas:

- Gradual ramp up with monitoring
- Rollback

- One type of deployment is if you are offering a new product or capability that you had not previously offered.
- For example, if you're offering a speech recognition service that you have not offered before, in this case, a common design pattern is to start up a small amount of traffic and then gradually ramp it up.
- A second common deployment use case is if there's something that's already being done by a person, but we would now like to use a learning algorithm to either automate or assist with that task.
- For example, if you have people in the factory inspecting smartphones scratches, but now you would like to use a learning algorithm to either assist or automate that task.
- The fact that people were previously doing this gives you a few more options for how you deploy. And you see shadow mode deployment takes advantage of this.
- And finally, a third common deployment case is if you've already been doing this task with a previous implementation of a machine learning system, but you now want to replace it with a better one.
- In these cases, **two recurring themes** you see are that you often want a gradual ramp up with monitoring. In other words, rather than sending tons of traffic to a maybe not fully proven learning algorithm, you may send it only a small amount of traffic and monitor it and then ramp up the percentage or amount of traffic.
- And the second idea you see a few times is rollback. Meaning that if for some reason the algorithm isn't working, it's nice if you can revert back to the previous system

### Visual inspection example

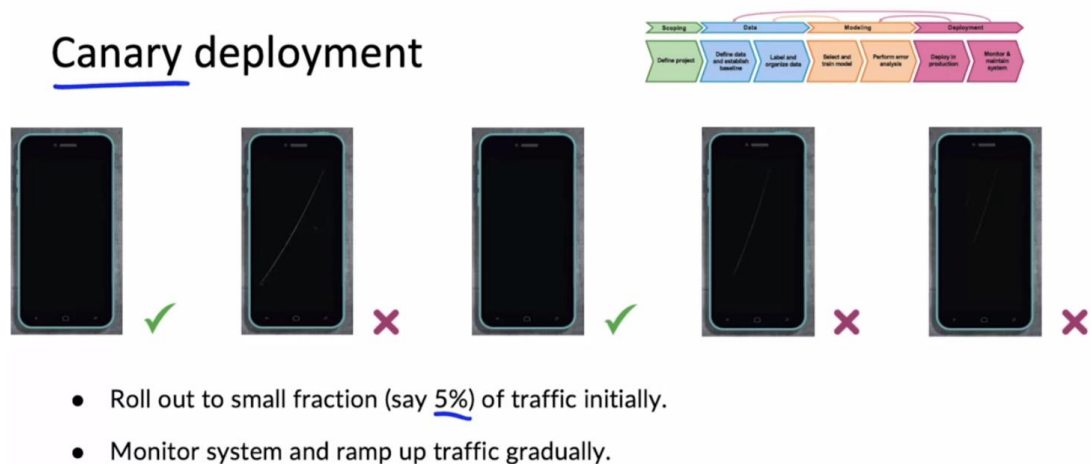
*shadow mode*



ML system shadows the human and runs in parallel.

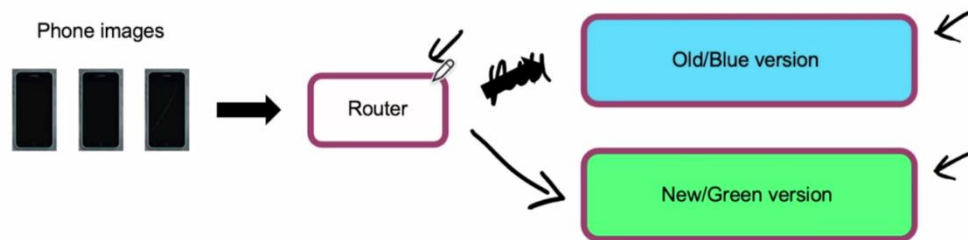
ML system's output not used for any decisions during this phase.

- When you have people initially doing a task, one common deployment pattern is to use **shadow mode deployment**. And what that means is that you will start by having a machine learning algorithm shadow the human inspector and running parallel with the human inspector.
- During this initial phase, the learning algorithms output is not used for any decision in the factory. So whatever the learning algorithm says, we're going to go the human judgment for now.
- So let's say for this smartphone (left image), the human says it is fine, no defect. The learning algorithm says it's fine. Maybe for this example (centre image) of a big stretch down the middle person says it's not okay and the learning algorithm agrees.
- And for this example (right image) with a smaller stretch, maybe the person says this is not okay, but the learning algorithm makes a mistake and actually thinks this is okay.
- The purpose of a shadow mode deployment is that allows you to gather data of **how the learning algorithm is performing and how that compares to the human judgment**.
- This allows you to verify if the learning algorithm's predictions are accurate, and therefore use that to decide whether or not to allow the learning algorithm to make some real decisions in the future.
- Using a shadow mode deployment can be a very effective way to let you verify the performance of a learning algorithm before letting them make any real decisions.



- When you are ready to let a learning algorithm start making real decisions, a common deployment pattern is to use a canary deployment.
- In a canary deployment you would roll out to a small fraction, maybe 5% (or even less of traffic initially) and start to let the algorithm making real decisions.
- But by running this on only a small percentage of the traffic, if the algorithm makes any mistakes it will affect only a small fraction of the traffic.
- And this gives you more of an opportunity to monitor the system and ramp up the percentage of traffic it gets only gradually and only when you have greater confidence in this performance.
- The phrase canary deployment is a reference to the English idiom, which refers to how coal miners used to use canaries to spot if there's a gas leak.
- With canary the deployment, hopefully this allows you to spot problems early on before there are maybe overly large consequences to a factory or other context in which you're deploying your learning algorithm.

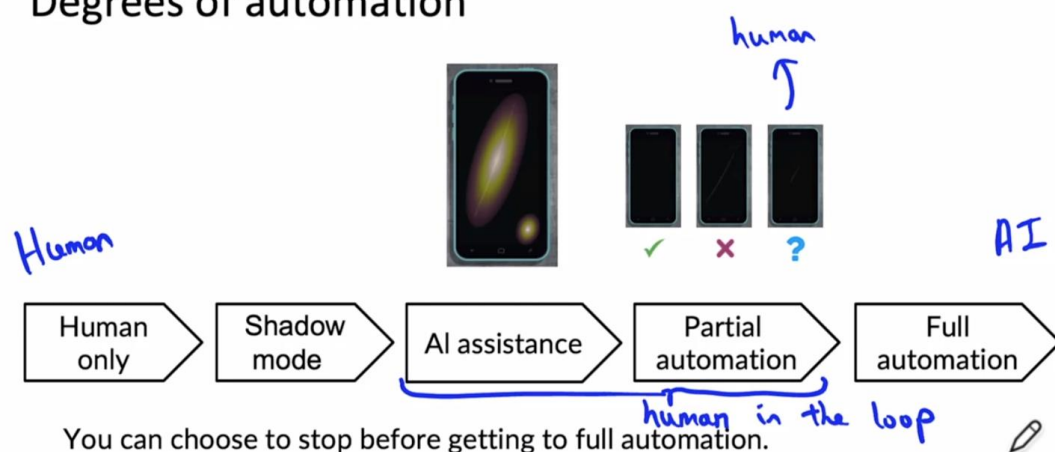
## Blue green deployment



Easy way to enable rollback

- Another deployment pattern that is sometimes used is a blue green deployment.
- Say you have a system, a camera software for collecting phone pictures in your factory. These phone images are sent to a piece of software that takes these images and routes them into some visual inspection system.
- In the terminology of a blue green deployments, the **old version of your software is called the blue version** and the new version, the learning algorithm you just implemented is called the green version.
- In a blue green deployment, what you do is have the router send images to the old (blue) version and have that make decisions. And then when you want to switch over to the new version, what you would do is have the router stop sending images to the old one and suddenly switch over to the new version.
- So the way the blue green deployment is implemented is you would have an old prediction service may be running on some sort of service. You will then spin up a new prediction service, the green version, and you would have the router suddenly switch the traffic over from the old one to the new one.
- The advantage of a blue green deployment is that there's an easy way to enable rollback. If something goes wrong, you can just very quickly have the router go back reconfigure their router to send traffic back to the old or the blue version, assuming that you kept your blue version of the prediction service running.
- In a typical implementation of a blue green deployment, people think of switching over the traffic 100% all at the same time. But of course you can also use a more gradual version where you slowly send traffic over.

## Degrees of automation



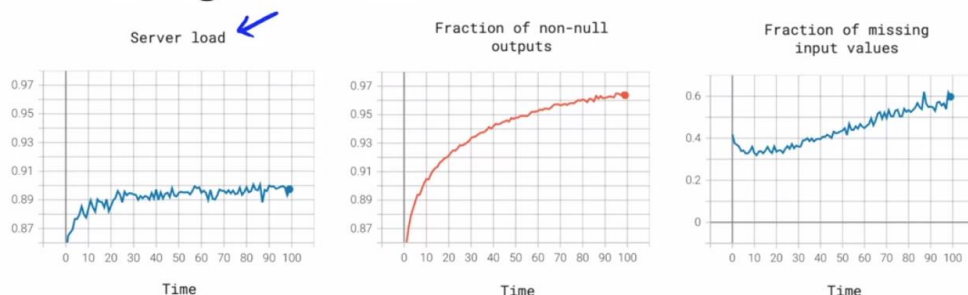
- One of the most useful frameworks I have found for thinking about how to deploy a system is to think about deployment not as a 0 / 1 (i.e. either deploy or not deploy), but instead to design a system thinking about what is the **appropriate degree of automation**.



- For example, in visual inspection of smartphones, one extreme would be if there's no automation, (human only system). Slightly more automated would be if your system is running a shadow mode, where the learning algorithms are giving predictions but are not actually used in the factory.
- A slightly greater degree of automation would be AI assistance where you have a human inspector making the decisions, but an AI system provides an user interface to highlight the regions where there's a scratch to help draw the person's attention to where it may be most useful for them to look. The user interface or UI design is critical for human assistance.
- An even greater degree of automation is partial automation. If the learning algorithm is sure/confident that the smartphone is fine or defective, then that's the final decision. But if the learning algorithm is not sure, in other words, if the learning algorithm prediction is not too confident, then we send this to a human to make the decision.
- So this would be partial automation, where if the learning algorithm is confident of its prediction, we go the learning algorithm. But for the small subset of images where the algorithm is not sure we send that to a human to get their judgment.
- And the human judgment can also be very valuable data to feedback to further train and improve the algorithm.
- I find that this partial automation is sometimes a very good design point for applications where the learning algorithms **performance isn't good enough for full automation**.
- And then of course beyond partial automation, there is **full automation** where we might have the learning algorithm make every single decision.
- So there is a spectrum of using only human decisions on the left, all the way to using only the AI system's decisions on the right. And many deployment applications will start from the left and gradually move to the right.
- You do not have to get all the way to full automation. You could choose to stop using AI assistance or partial automation or you could choose to go to full automation depending on the performance of your system and the needs of the application.
- On this spectrum both AI assistance and partial automation are examples of **human in the loop deployments**.
- A lot of consumer software Internet businesses have to use full automation because it's just not feasible to someone on the back end doing some work every time someone does a web search or does the product search.
- But outside consumer software Internet, for example, inspecting things and factories, they are actually many applications where the best design point maybe a human in the loop deployments rather than a full automation deployment.

## Monitoring

### Monitoring dashboard



- Brainstorm the things that could go wrong.
- Brainstorm a few statistics/metrics that will detect the problem.
- It is ok to use many metrics initially and gradually remove the ones you find not useful.



- How can you monitor a machine learning system to make sure that it is meeting your performance expectations? The most common way to monitor a machine learning system is to use a dashboard to track how it is doing over time.
- Depending on your application, your dashboards may monitor different metrics. For example, you may have one dashboard to monitor the server load, or a different dashboards to monitor the fraction of non-null outputs. Sometimes a speech recognition system output is null when the things that users didn't say anything.
- If this changes dramatically over time, it may be an indication that something is wrong
- One common one I've seen for a lot of structured data task is monitoring the fraction of missing input
- When you're trying to decide what to monitor, my recommendation is that you sit down with your team and brainstorm all the things that could possibly go wrong. Then you want to know about if something does go wrong. For all the things that could go wrong, brainstorm a few statistics or a few metrics that will detect that problem.
- For example, if you're worried about user traffic spiking, causing the service to become overloaded, then server loads maybe one metric, you could track and so on for the other examples

## Examples of metrics to track

### Software metrics:

Memory, compute, latency, throughput, server load

### Input metrics:

x

Avg input length  
Avg input volume  
Num missing values  
Avg image brightness

### Output metrics:

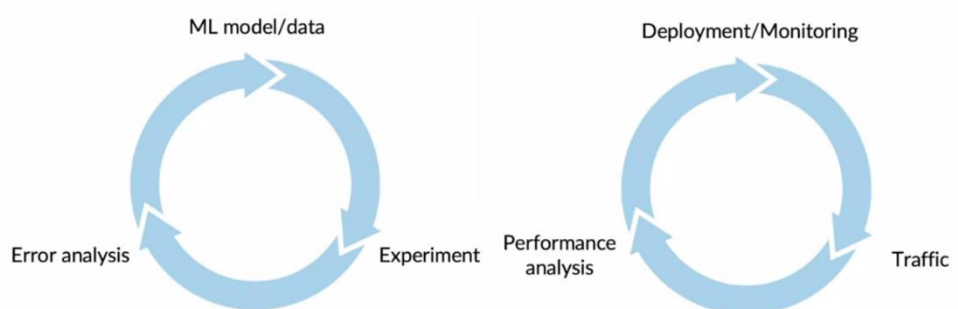
y

# times return " " (null)  
# times user redoes search  
# times user switches to typing  
CTR

- Here are some examples of metrics our views or I've seen others use on a variety of projects. First are the software metrics, such as memory, compute, latency, throughput, server load i.e. things that help you monitor the health of your software implementation of the prediction service or other pieces of software around your learning algorithm.
- Many MLOps tools are already tracking these software metrics.
- In addition to the software metrics, I would often choose other metrics that help monitor the statistical health or the performance of the learning algorithm. Broadly, there are two types of metrics you might brainstorm around.
- One is input metrics, which measure whether your input distribution **X** has changed
- For example, if you are building a speech recognition system, you might monitor the average input length in seconds of the length for the audio clip fed to your system.
- If these change for some reason, which might be something you'll want to take a look at just to make sure it hasn't hurt the performance of your algorithm.
- The number (or percentage of) missing values is a very common metric (e.g. when using structured data, some of which may have missing values)
- For the manufacturing visual inspection example, you might monitor average image brightness if you think that lighting conditions could change, and you want to make sure you know if it does, so you can brainstorm different metrics to see if your input distribution **x** might have changed.
- A second set of metrics that help you understand if the algorithm is performing well are **output metrics**

- Such as, how often does your speech recognition system return null, the empty string, because the things the user doesn't say anything, or if you have built a speech recognition system for web search using voice, you might decide to see how often does the user do two very quick searches in a row with substantially the same input.
- That might be a sign that you misrecognize their query the first time round. It's an imperfect signal but you could try this metric and see if it helps.
- Or you could monitor the number of times the user first try to use the speech system and then switches over to typing, that could be a sign that the user got frustrated or gave up on your speech system and could indicate degrading performance.
- Because input and output metrics are application specific, most MLOps tools will need to be configured specifically to track the input and output metrics for your application.

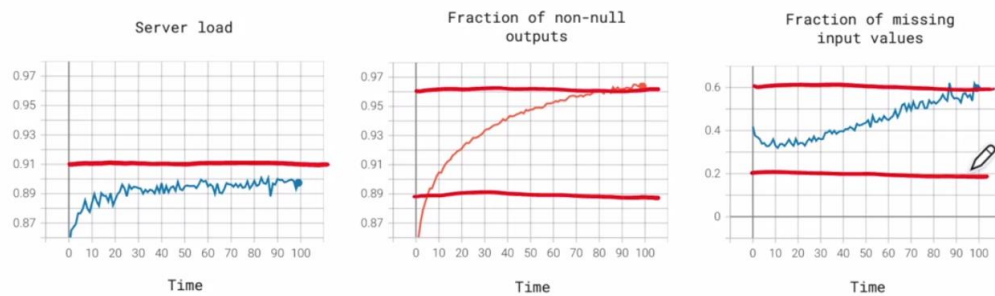
## Just as ML modeling is iterative, so is deployment



Iterative process to choose the right set of metrics to monitor.

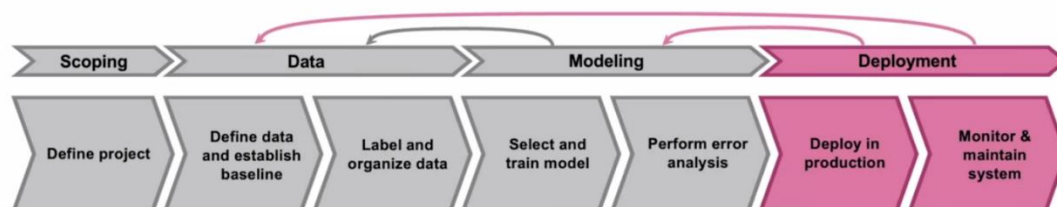
- You may already know that machine learning modelling is a highly iterative process, so as deployment.
- I encourage you to think of deployments as an iterative process as well. When you get your first deployments up and running and put in place a set of monitoring dashboards.
- But that's only the start of this iterative process. A running system allows you to get real user data or real traffic. It is by seeing how the learning algorithm performs on real data on real traffic that allows you to do performance analysis, and this in turn helps you to update your deployment and to keep on monitoring your system.
- It usually takes a few tries to converge to the right set of metrics to monitor. It's not uncommon for you to deploy machine learning system with an initial set of metrics, only to run the system for a few weeks and realize something could go wrong that you hadn't thought of before, and you will need to pick a new metric to monitor.
- Or for you to have some metric that you monitor for a few weeks and then decide they hardly ever change, then we can get rid of that metric in favour of focusing attention on something else.

# Monitoring dashboard



- Set thresholds for alarms
  - Adapt metrics and thresholds over time
- After you've chosen a set of metrics to monitor, a common practice would be to set thresholds for alarms. You may decide based on this set, if the server load ever goes above 0.91, it may trigger an alarm or a notification to let you or the team know to see if there's a problem and maybe spin up some more servers.
  - It is okay if you adapt the metrics and the thresholds over time to make sure that they are flagging to you the most relevant cases of concern. If something goes wrong with your learning algorithm, if is a software issue such as server load is too high, then that may require changing the software implementation,
  - If it is a performance or accuracy problem associated with the accuracy of the learning algorithm, then you may need to update your model.
  - That's why many machine learning models need a little bit of maintenance or retraining over time.

## Model maintenance

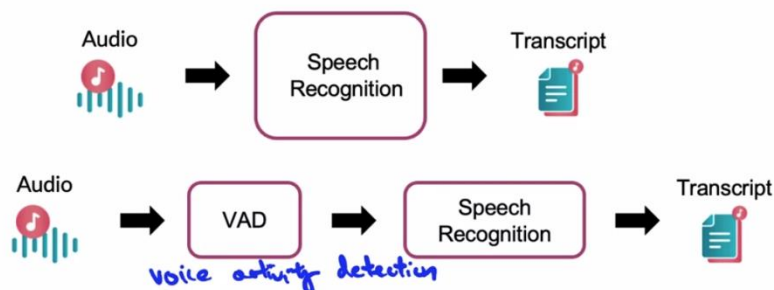


- Manual retraining ←
  - Automatic retraining ←
- When a model needs to be updated, you can either retrain it manually (together with the engineers), or you could also put in place a system where there is automatic retraining.
  - Today, manual retraining is far more common than automatically training. For many applications, developers are reluctant to learning algorithm be fully automatic in terms of deciding to retrain and pushing new model to production, but there are some applications, especially in consumer software Internet, where automatically training does happen.
  - But the key takeaways are that it is only by monitoring the system that you can spot if there may be a problem that may cause you to go back to perform a deeper error analysis, or that may cause you to go back to get more data with which you can update your model so as to maintain or improve your system's performance.

## Pipeline Monitoring

- Many AI systems are not just a single machine learning model running a prediction service, but instead involves a pipeline of multiple steps.

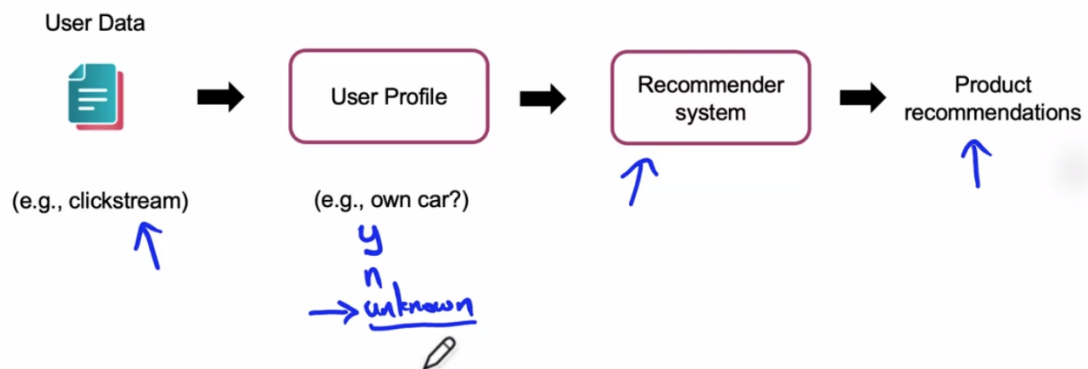
### Speech recognition example



Some cellphones might have VAD clip audio differently, leading to degraded performance

- Let's continue with our speech recognition example, you've seen how a speech recognition system may take as input audio and I'll put a transcript.
- The way that speech recognition is typically implemented is a slightly more complex pipeline, where the audio is fed to a module called a VAD (voice activity detection) module, whose job it is to see if anyone is speaking. And only if the VAD module thinks someone is speaking, then does it bother to pass the audio on to a speech recognition system whose job it is to then generate the transcript.
- We use a VAD module because if your speech recognition system runs in the cloud, you don't want to stream more bandwidth than you have to into your cloud server.
- And so the VAD module looks at the long stream of audio on your cell phone and clips or shortens the audio to just the part where someone is talking and streams only that to the cloud server to perform the speech recognition.
- So this is an example of a machine learning pipeline where there is one step done by a learning algorithm to decide if someone is talking or not, and then the second step, also done by a learning algorithm, to generate the text transcript.
- When you have two such modules working together, changes to the first module may affect the performance of the second module as well.
- For example, let's say that because of the way a new cell phone's microphone works, the VAD module ends up clipping the audio differently. Maybe it leaves more silence at the start or end or less silence at the start or end.
- This will cause the speech recognition systems input to change, and that could cause degraded performance of the speech recognition system.

## User profile example



- Let's look an example involving user profiles. I've used the data such as clickstream data showing what users are clicking on. And this can be used to build a user profile that tries to capture key attributes or key characteristics of a user.
- For example, I once built user profiles that would try to expect many attributes of users including whether or not the user seemed to own a car. Because this would help us decide if it was worth trying to offer car insurance office to that user.
- And so whether the user owns a car could be yes or no or unknown. The typical way that the user profile is built is with a learning algorithm to try to predict if this user of the car.
- This type of user profile, which can have a very long list of predicted attributes, can then be fed to recommend a system.
- Another learning algorithm that then takes this understanding of the user to try to generate product recommendations.
- Now, if something about the click stream data changes, maybe the input distribution changes, then over time we lose our ability to figure out if a user owns a car, where the percentage of the unknown label here may go up.
- And because the user profiles change, the input to the recommender system now changes, and this might affect the quality of the product recommendations.

## Metrics to monitor

### Monitor

- Software metrics
- Input metrics
- Output metrics

### How quickly do they change?

- User data generally has slower drift.
- Enterprise data (B2B applications) can shift fast.

- So when building these complex machine learning pipelines, which can have ML-based learning based components or non-ML based components throughout the pipeline.
- I find it useful to brainstorm metrics to monitor that can detect changes including concept drift or data driven or both, and multiple stages of the pipeline.

- Metrics to monitor include software metrics for each of the components in the pipeline, or perhaps for the overall pipeline as a whole, as well as input metrics and potentially output metrics for each of the components of the pipeline.
- And by brainstorming metrics associated with individual components of the pipeline as well.
- The principle that you saw in the last video of brainstorming about all the things that could go wrong, including things that could go wrong with individual components of the pipeline and design metrics to track those, still applies. Only now you're looking at multiple components in the pipeline.
- Finally, how quickly does data change? The rate at which data changes is very problem dependent.
- For example, let's say we built a face recognition system. The rate at which people's appearances changes usually isn't that fast. People's hairstyles and clothing does change with fashion changes. And as cameras get better, we've been getting higher and higher resolution pictures of people over time. But for the most part, people's appearances don't change that much.
- There are sometimes where things can change very quickly as well, such as if a factory gets a new batch of material for how they make cell phones, so all the cell phones change in appearance. So some applications will have data that changes over the time scale of months or even years.
- Some applications with data that could suddenly change in a matter of minutes. Speaking in very broad generalities, I find that on average, **user data generally changes relatively slowly**. If you run a consumer facing business with a very large numbers of users, then it is quite rare for millions of users to all suddenly change their behaviour all at the same time.
- There are a few exceptions, of course, COVID-19 being one of them were a shock to society actually cause a lot of people's behaviour that all change at the same time.
- So there are exceptions, but on average, if you have a very large group of users, there are only a few forces that can simultaneously change the behaviour of a lot of people at the same time.
- In contrast, if you work on a B2B or business to business application, I find an **enterprise data or business data can shift quite quickly**. Because the factory making cell phones may suddenly decide to use a new coating for the cell phones, and suddenly the entire dataset changes because the cell phones suddenly all look different. Or sometimes if the CEO of that company decides to change the way that business operates, all of that data can shift very quickly.
- I know that these two bullets are speaking in generalities and there are certain exceptions to both of these. But maybe this will give you a way of thinking about how quickly your data is likely to change or not change.

## Labs

- <https://github.com/https-deeplearning-ai/MLEP-public/tree/main/course1/week1-ungraded-lab>

## Reading References

- <https://towardsdatascience.com/machine-learning-in-production-why-you-should-care-about-data-and-concept-drift-d96d0bc907fb>
- <https://christophergs.com/machine%20learning/2020/03/14/how-to-monitor-machine-learning-models/http://arxiv.org/abs/2011.09926>
- <https://papers.nips.cc/paper/2015/file/86df7dcfd896fcac2674f757a2463eba-Paper.pdf>
- <http://arxiv.org/abs/2010.02013>