

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 3: Data Definition and Baseline

Contents

Week 3: Data Definition and Baseline	1
Why is data definition hard?	1
More label ambiguity examples	3
Major types of data problems	3
Small data and label consistency	8
Improving label consistency	10
Human level performance (HLP)	13
Raising HLP	15
Obtaining data	17
Data pipeline	20
Meta-data, data provenance and lineage	22
Balanced train/dev/test splits	23
What is scoping	24
Scoping process	25
Diligence on feasibility and value	27
Diligence on value	30
Milestones and resourcing	32
References	32

Why is data definition hard?

- I'm going to use the example of detecting iguanas

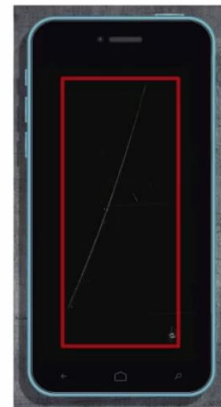
Iguana detection example



Labeling instructions: "Use bounding boxes to indicate the position of iguanas"

- Three labellers can come up with these three different ways of labelling iguanas, and maybe any one of these is actually fine. I would prefer the top two rather than the 3rd one.
- But what is not fine is if 1/3 of your labellers uses the 1st convention and 1/3 the 2nd, and 1/3 the 3rd labelling convention
- Then your labels will be inconsistent, and this is confusing to the learning algorithm.

Phone defect detection



- If you ask a labeller to use bounding boxes to indicate significant defects, maybe one labeller will look and then go, "Well, clearly the scratch is the most significant defect."
- A 2nd labeller may look at the phone and say, "There are actually two significant defects. There's a big scratch, and then there's that small pit mark. But then a 3rd labeller may look at this and say, well, here's a bounding box that shows you where all the defects are."
- Between these three labels, probably the one in the middle would work the best. But this is a very typical example of inconsistent labelling that you will get from a labelling process, with just slightly ambiguous labelling instructions
- Many machine learning researchers and engineers had started off downloading data off the Internet to experiment with models, so using data prepared by someone else. Nothing at all wrong with that, but for many practical applications, the way you prepare your data sets will have a huge impact on the success of your machine learning projects.

More label ambiguity examples

Speech recognition example



"Um, nearest gas station"

"Umm, nearest gas station"

"Nearest gas station [unintelligible]"

- Given this audio clip, it sounds like someone was standing on a busy road side asking for the nearest gas station and then a car drove past.
- There are many combinatorial ways to transcribe this. With one M or two M's, comma or ellipses, whether to write unintelligible at the end of this.
- Being able to standardize on one convention will help your speech recognition algorithm.

Major types of data problems

User ID merge example

	Job Board (website)	Resume chat (app)
Email	nova@deeplearning.ai	nova@chatapp.com
First Name	Nova	Nova
Last Name	Ng	Ng
Address	1234 Jane Way	?
State	CA	?
Zip	94304	94304

Handwritten notes:

- is it a bot/spam account?
- fraudulent transaction?
- looking for job?

Legend:

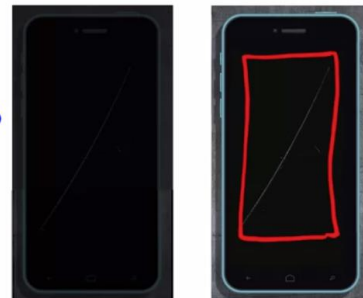
- 1 if same
- 0 if different

- A common application in many large companies is **user ID merge**. That's when you have multiple data records that you think correspond to the same person, and you want to merge these user data records together.
- For example, say you run a website that offers online listings of jobs. This may be one data record that you have from one of your registered users with the email, first name, last name and address.
- Now, say your company acquires a second company that runs a mobile app that allows people to login, to chat and get advice from each other about their resumes.
- It seems synergistic for your business, and you now have a different database of users.
- Given this data record and this one (shown in slide), do you think these two are the same person?

- One approach to the User ID merge problem, is the use of supervised learning algorithm that takes user data records as input and outputs either one or zero based on whether it thinks these two are actually the same human being.
- If you have a way to get ground truth records, such as if a handful of users are willing to explicitly link the two accounts, then that could be a good set of labelled examples to train an algorithm.
- But if you don't have such a ground truth set of data, what many companies have done is ask human labourers, sometimes a product management team, to just manually look at some pairs of records that have been filtered to have maybe similar names or similar ZIP codes, and then use human judgment to determine if these two records appear to be the same person.
- Whether these two records really refer to the same person can be genuinely ambiguous.
- They may and they may not be different people
- If there's a way to just get them to label the data a little more consistently, even when the ground truth is ambiguous, it can still help the performance of your learning algorithm.
- User ID merging is a very common function in many companies, so let me just ask you to please do this only in ways that are respectful of the users data and their privacy, and only if you're using the data in a way that is consistent with what they have given you permission for.
- A few other examples from structured data. If you are trying to use the learning algorithm to look at the user account predict is it a bot or a spam account. Or if you look at an online purchase and check if this is a fraudulent transaction. Sometimes that too is ambiguous.
- In the face of potentially very important and valuable prediction tasks like these, the ground truth can be ambiguous.
- If you ask people to take their best guess at the ground truth label for tasks like these, giving labelling instructions that results in more consistent and less noisy and less random labels will improve the performance of your learning algorithm.

Data definition questions

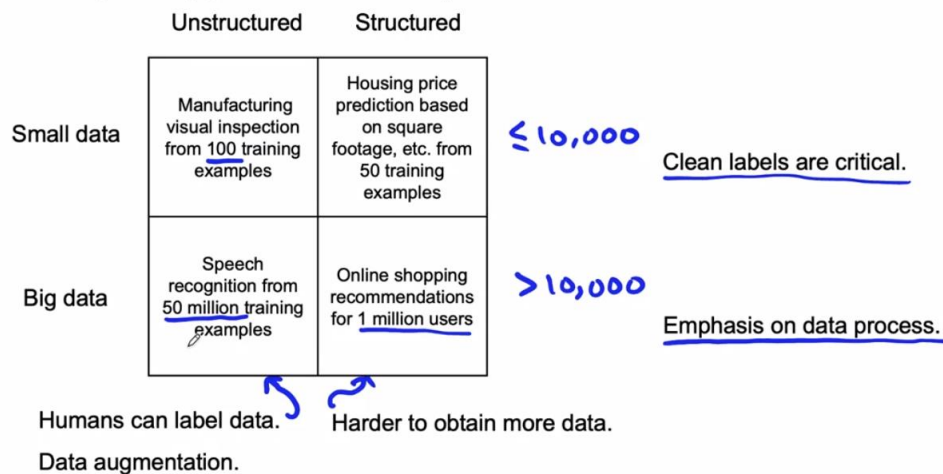
- What is the input x ?
 - Lighting? Contrast? Resolution?
 - What features need to be included?
- What is the target label y ?



- When defining the data for your learning algorithm, here are some important questions.
- First, what is the input x ?
- For example, if you are trying to detect defects on smart phones, for the pictures you're taking, is the lighting good enough? Is the camera contrast good enough? Is the camera resolution good enough?
- If you find that you have a bunch of pictures like these, which are so dark that it's hard even for a person to see what's going on, the right thing to do may **not** be to take this input x and just label it.
- It may be better to go to the factory and politely request to improve the lighting, because it is only with this better image quality that the labeller can better see scratches like this and label them.
- If your sensor or your imaging solution or your audio recording solution is not good enough, the best thing you could do is recognize that if a person can't look at the input and tell us what's going on, then improving the quality of your sensor or improving the quality of the input x is the important first step to ensuring your learning algorithm has reasonable performance.
- For structured data problems, defining what features to include can have a huge impact on your learning algorithm's performance.

- For example, for user ID merge, if you have a way of getting the user's location, even a rough GPS location (provided you have permission from the user to use that), it can be a very useful tool for deciding whether two user accounts actually belong to the same person.
- In addition to defining the input x , you also have to figure out what should be the target label y .
- As you've seen from the preceding examples, one key question is, how can we ensure labels give consistent labels? In the last video and this video, you saw a variety of problems with the labels being ambiguous or in some cases, the input x not being sufficiently informative (e.g. images too dark)

Major types of data problems



- It turns out that the best practices for organizing data for one type can be quite different than the best practices for totally different types. Let's look at this 2 x 2 grid.
- One axis will be whether your machine learning problem uses unstructured data or structured data.
- I found that the best practices for these are very different, mainly because humans are great at processing unstructured data, the images and audio and text, and not as good at processing structured data like database records.
- The second axis is the size of your data set. There is no precise definition of what exactly is small and what is large. But I'm going to use as a slightly arbitrary threshold, whether you have over **10,000** examples or not.
- I chose the number **10,000** because that is roughly the size beyond which it becomes quite painful to examine every single example yourself.
- Let's look at some examples. If you are training a manufacturing visual inspection from just 100 examples of stretch phones, that's unstructured data with a pretty small data set. If you are trying to predict housing prices based on the size of the halls and other features, from just 52 examples, then there's a structured data set with a small dataset
- If you are carrying out speech recognition from 50 million train examples, that's unstructured data with a large dataset
- If you are doing online shopping recommendations and you have a million users in your database, then that's a structured problem with relatively large amount of data.
- For manufacturing vision inspection, you can use **data augmentation** to generate more pictures of smart films or for speech recognition. Data augmentation can help you synthesize audio clips with different background noises too.
- In contrast for structured data problems, it can be harder to obtain more data and also harder to use data augmentation. It's hard to synthesize new users that don't really exist, and it's also harder (may or may not be possible) to get humans to label the data.
- So I find that the best practices for unstructured versus structured data are quite different
- When you have a relatively small data set, having clean labels is critical. If you have 100 training examples, then if just one of the examples is mislabelled, that's 1% of your data set.

- And because the data set is small enough for you or a small team to go through it efficiently, it may well be worth the while to go through those 100 examples. It is important to make sure that every one of those examples is labelled in a clean and consistent way, according to a consistent labelling standard
- In contrast, if you have a million data points, it can be harder (almost impossible) to manually go through every example.
- Having clean labels is still very helpful, since clean labels is better than non-clean ones.
- But because of the difficulty of having the team manually go through each example, the emphasis is on **data processes**, in terms of how you collect the data, install the data, the labelling instructions you write for a large team of crowdsourced labellers.

Unstructured vs. structured data

Unstructured data

- May or may not have huge collection of unlabeled examples x .
- Humans can label more data.
- Data augmentation more likely to be helpful.

Structured data

- May be more difficult to obtain more data.
- Human labeling may not be possible (with some exceptions).

- For unstructured data problems, you may or may not have a huge collection of unlabelled examples x .
- Maybe in your factory, you actually took thousands of images of smartphones, but you just haven't bothered to label all of them yet. This is also common in the self-driving car industry, where many self-driving car companies have collected tons of images of cars driving around, but just have not yet caught in that data labelled.
- For these unstructured data problems, you can get more data by asking humans to just label more of the unlabelled data
- And as we have already mentioned, data augmentation can also be helpful.
- For structured data problems is usually harder to obtain more data because you only have so many users that you can collect data from.
- And human labelling on average is also harder, although there are some exceptions, such as where you saw that we could try to ask people to label examples for user ID merge.
- But in many cases where we ask humans to label structure data, even when it is completely worthwhile to ask people to try to label if two records are the same person, there's likely to be a more ambiguity where even the human labeller can find it hard to be sure what the correct label is.

Small data vs. big data

Small data

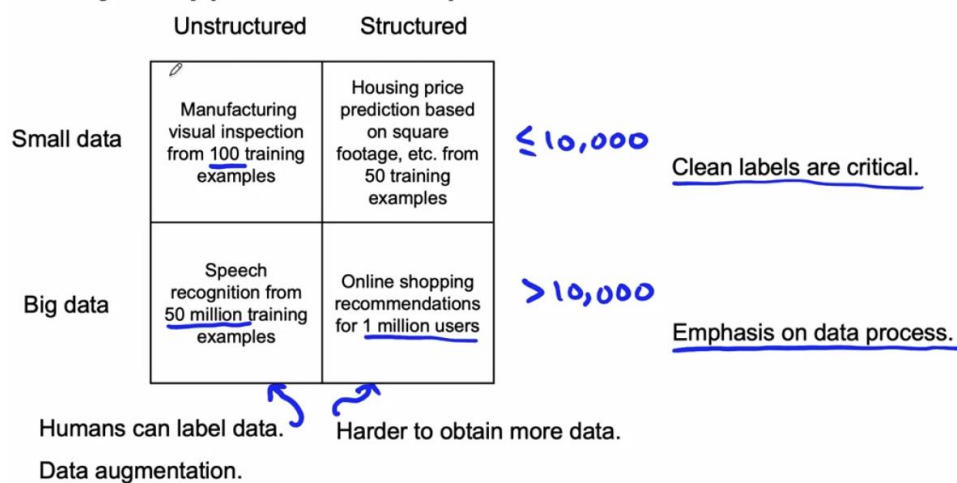
- Clean labels are critical.
- Can manually look through dataset and fix labels.
- Can get all the labelers to talk to each other.

Big data

- Emphasis data process.

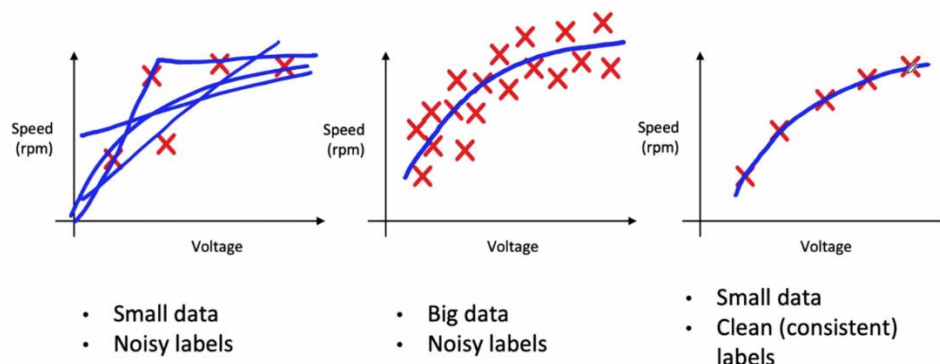
- Let's look at small versus big data, where I used to a slightly arbitrary threshold of whether you have more or less than 10,000 training examples.
- For small data sets, clean labels are critical and the data set may be small enough for you to manually look through the entire data set and fix any inconsistent labels.
- Furthermore, the labelling team is probably not that large, so it is easier to have them talk to each other and hash out and agree on one labelling convention.
- For the very large data sets, the emphasis has to be on data process. And if you have a 100 labellers or even more, it's just harder to get 100 people into a room to all talk to each other and hash out the process.
- And so you might have to rely on a smaller team to establish a consistent label definition and then share that definition with the larger team of labellers and ask them to all implement the same process.

Major types of data problems



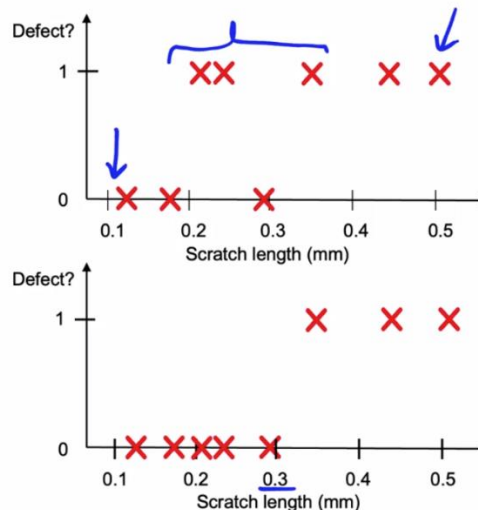
- I want to leave you with one last thought, which is that I found this categorization of problems into unstructured versus structured, small versus big data to be helpful
- If you are working on a problem from one of these four quadrants, taking advice from someone that has worked on problems in the same quadrant will probably be more useful than advice from someone that's worked in a different quadrant.
- I found also in hiring machine learning engineers, someone that's worked in the same quadrant as the problem I'm trying to solve will usually be able to adapt more quickly to working on other problems in that quadrant.
- Because the instincts and decisions are more similar within one quadrant than if you shift to a totally different quadrant.
- I've sometimes heard people give advice like, if you are building a computer vision system, always get at least 1000 labelled examples.
- I think people that give advice like that are well meaning and I appreciate that they're trying to give good advice, but I found that advice to not really be useful for all problems.
- Machine learning is very diverse and it's hard to find one size fits all advice like that.
- I've seen computer vision problems built with 100 examples or even systems built with 100 million examples.
- So if you are looking for advice on a machine learning project, try to find someone that's worked in the same quadrant as the problem you are trying to solve.

Why label consistency is important



- In a small dataset, having clean and consistent labels is especially important.
- One of the things I used to do is use machine learning to fly helicopters. What you might want to do is input the voltage applied to the motor or helicopter rotor and predict what the speed of the rotor is.
- So let's say you have a data set that looks like this, where you have five examples.
- From the small dataset, the output Y is pretty noisy, so it is difficult to know the function to use to map voltage to the rotor speed in rpm.
- Maybe it should be a straight line, or maybe a curve like that. It is really hard to tell when you have a small data set, especially with noisy labels
- Now, if you had data which is equally noisy as the one on the left, but you have a lot more data. The learning algorithm can then average over the noisy data and you can now fit a function pretty confidently
- A lot of AI had recently grown up in large consumer Internet companies which may have 100 million users or billion users and thus with very large data sets.
- And so, I think some of the practices for how to deal with small data sets have not been emphasized as much as would be needed to tackle problems where you don't have
- 100 million examples, but only 1000 or even fewer.
- So to me, the interesting case is what if you still have a small data set?
- On the right, we see five examples with clean and consistent labels. In this case you can pretty confidently fit a function through your data and with only five examples.
- You can build a pretty good model for predicting speed as a function of the input voltage of trained computer vision systems with just 30 images and can work just fine.
- The key is usually to make sure that the labels are clean and consistent.

Phone defect example



- Let's take a look at another example of phone defect inspection,
- If labelling instructions are initially unclear, then labellers will label images inconsistently.
- There's might be this region of ambiguity where different inspectors will label different scratches with a length between 0.2mm and 0.4mm in slightly inconsistent ways.
- So one solution to this would be to get a lot more pictures of phones and scratches, see what the inspectors do, and then maybe eventually we can train a neural network that can figure out from the images what is and what isn't a scratch on **average**.
- Maybe that approach could work, but it'd be a lot of work and require collecting a lot of images.
- I found that it can be more fruitful to ask the inspectors to sit down and just try to reach agreement on what is the size of scratch to define as a defect
- So in this example, if the labellers can agree that the point of transition from where little scratches becomes a defect is a length of 0.3mm, then the way they label the images (i.e. set bounding boxes) becomes much more consistent.
- And it becomes much easier for learning algorithms to take input images like this and consistently decide whether something is a scratch or a defect.
- Higher consistency in image labelling allow your learning algorithms to achieve higher accuracy, even when your data set isn't that big.

Big data problems can have small data challenges too

Problems with a large dataset but where there's a long tail of rare events in the input will have small data challenges too.

- Web search
- Self-driving cars ←
- Product recommendation systems ←

- Big data problems can have small data challenges too, specifically problems of the large data set where there's a long tail of rare events in the input
- For example, the large web search engine companies all have very large datasets of web search queries, but many queries are actually very rare. So the amount of click stream data for the rare queries is actually small
- Take self-driving cars. Self-driving car companies tend to have very large data sets, collected from driving hundreds of thousands or millions of hours or more

- But there are rare occurrences that are critical to get right to make sure a self-driving car is safe, such as the very rare occurrence of a young child running across the highway, or the very rare occurrence of a truck parked across the highway.
- So even if a self-driving car has a very large data set, the number of examples that may have of these rare events is actually very small. Ensuring label consistency in terms of how these rare events are labelled is still very helpful for improving self-driving cars.
- For product recommendation systems, if you have a catalogue of hundreds of thousands, or millions or more items, then you will have a lot of products where the number sold for that item is quite small.
- The amount of data you have of users interacting with the items in the long tail is actually small, so getting the data clean and consistent will help your learning algorithms.
- So when you have a small dataset label consistency is critical. Even when you have a big data set, label consistency can be very important.
- It's just that it is easier on average to get label consistency on smaller datasets than on very large ones.

Improving label consistency

Improving label consistency

- Have multiple labelers label same example.
 - When there is disagreement, have MLE, subject matter expert (SME) and/or labelers discuss definition of y to reach agreement.
 - If labelers believe that x doesn't contain enough information, consider changing x .
 - Iterate until it is hard to significantly increase agreement.
-
- Let's take a look at some ways to improve the consistency of your labels
 - If you are worried about labels being inconsistent, find a few examples and have multiple labellers label those same examples.
 - You can also have the same labeller label an example, let them take a break, and then come back and re-label it and see if they're even consistent with themselves.
 - When you find that there's disagreements, have the people responsible for labelling to discuss together what they think should be a more consistent definition of the label y ,
 - Have them try to reach an agreement, and ideally also document and write down that agreement
 - This definition of y can then become an updated set of labelling instructions, which they can then use to label new data or to re-label old data.
 - During this discussion, in some cases the labellers will come back and say they don't think the input x has enough information. If that's the case, consider changing the input x . For example, when some pictures of phones may be so dark that we couldn't even tell what was going on,
 - That was a sign that we should consider increasing the lighting under which the pictures were taken. But of course, I know this isn't always possible, but sometimes this can be a big help.
 - All this is an iterative process, so after improving the label instructions, you will then ask the team to label more data.
 - If you think there are still disagreements, then repeat the whole process of having multiple labellers label the same example, resolve the disagreement and so on.

Examples

- Standardize labels

"Um, nearest gas station"

"Umm, nearest gas station"

"Nearest gas station [unintelligible]"



"Um, nearest gas station"

- Merge classes



- One common outcome of this type of exercise is to standardize the definition of labels.
- Between these ways of labelling the audio clip you heard on the earlier video, perhaps the labellers will standardize on this as the convention. This makes the data more consistent.
- Another common decision that I've seen come out of a process like this is merging classes.
- If the definition between what constitutes a deep scratch versus a shallow scratch is unclear, then you end up with labellers inconsistently labelling things as deep versus shallow scratches.
- Sometimes the factory does really need to distinguish between deep versus shallow scratches.
- But sometimes I found is that you don't really need to distinguish between these two classes. If so, you can instead merge the two classes into a single class (e.g. all label as the scratch class), and this gets rid of all of the inconsistencies
- Merging classes isn't always applicable, but when it is, it simplifies the task for the learning algorithm.

Have a class/label to capture uncertainty

- Defect: 0 or 1



Alternative: 0, Borderline, 1

- Unintelligible audio



"nearest go"

"nearest grocery"

"nearest [unintelligible]"

- One of the technique I've used is to create a new class/label to capture uncertainty.
- For example, let's say you asked labellers to label phones as defective or not based on the length of the scratch.
- Maybe everyone agrees that the giant scratch is a defect, a tiny scratch is not a defect, but they don't agree on what's in between.

- Here's another option, which is to create a new class where you now have three labels, where ambiguous (genuinely borderline) examples are placed in a new borderline class.
- This helps to potentially improve labelling consistency.
- Let me use speech illustration to illustrate this further. Given this audio clip, I really can't tell what they said. If forced to label, some labellers would transcribe it as "Nearly go." while some will maybe transcribe as "Nearest grocery,"
- It's very difficult to get to consistency because the audio clip is genuinely ambiguous.
- To improve labelling consistency, it may be better to create a new tag called the unintelligible tag, and just ask everyone to label the ambiguous clip as unintelligible.
- This can result in more consistent labels than if we were to ask everyone to guess what they heard when it really is unintelligible.

Small data vs. big data (unstructured data)

Small data

- Usually small number of labelers.
- Can ask labelers to discuss specific labels.

Big data

- Get to consistent definition with a small group.
- Then send labeling instructions to labelers.
- Can consider having multiple labelers label every example and using voting }
or consensus labels to increase accuracy.

- Let me wrap up with some suggestions for working with small versus big datasets to improve label consistency.
- For small datasets there's usually a small number of labellers.
- So when you find an inconsistency, you can ask the labellers to sit down and discuss a specific image or a specific audio clip, and try to drive to an agreement.
- For big datasets, it would be more common to try to get to consistent definitions with a small group, and then send the labelling instructions to a larger group of labellers.
- One other technique that is commonly used, but I think overused in my opinion, is that you can have multiple labellers label every example, and then let them vote.
- Voting is sometimes called **consensus labelling**, and is used in order to increase accuracy.
- I find that this type of voting mechanism technique, it can work, but it's probably over used in machine learning today.
- Where what I've seen a lot of teams do is have inconsistent labelling instructions, and then try to have a lot of labellers and then perform voting to try to make it more consistent.
- But before resorting to this, which I do use but more of a last resort, I would first try to get to more consistent label definitions, to try to make the individual labellers' choices less noisy in the first place. This is opposed to taking a lot of noisy data and then try to use voting to reduce the noise.
- One of the gaps I see in the machine learning world today is that there's still a lack of tools to carry out this type of process more consistently and repeatedly.

Human level performance (HLP)

Why measure HLP?

Estimate Bayes error / irreducible error to help with error analysis and prioritization.

Ground Truth Label	Inspector
1	1 ✓
1	0 ✗
1	1 ✓
0	0 ✓
0	0 ✓
0	1 ✗

↑ Human?

99.0%

66.7% accuracy

- One of the most important users of measuring Human Level Performance or HLP is to estimate based error or irreducible error, especially on unstructured data tasks in order to help with their analysis and prioritization and establish what might be possible.
- Take for example visual inspection tasks. I have gotten requests from business owners asking me to build a system that's 99% accurate or maybe even 99.9% accurate.
- One way to establish what might be possible would be to take a data set and look at the Ground Truth Data.
- Say you have six examples with the ground truth labels (see slide), and you ask human inspector to label the same data blinded to the ground truth labels, and see what they come up with
- If the result is what is seen above in the slide, where the inspector agreed to the ground truth for just four of the six examples, then the HLP is 66.7%.
- And so this would let you go back to the business owner and say look, even your inspector got only 66.7% accuracy. How can you expect me to get 99% accuracy?
- So HLP is useful for establishing a baseline in terms of what might be possible.
- There's a question that is not often asked, which is what exactly this ground truth label is?
- We should think whether we are really measuring what is possible, or are we just measuring how well two different people happen to agree with each other (since the ground truth label itself is determined by a human too)

Other uses of HLP

- In academia, establish and beat a respectable benchmark to support publication.
- Business or product owner asks for 99% accuracy. HLP helps establish a more reasonable target.
- "Prove" the ML system is superior to humans doing the job and thus the business or product owner should adopt it.

✗ Use with caution

- In academia, HLP is often used as a respectable benchmark. And so when you establish that people are only 92% accurate (e.g. on a speech recognition data set), and if you can beat human level performance, that helps you to prove that the learning algorithm is doing something hard
- I'm not saying this is a great use of HLP, but in academia, showing you can beat HLP maybe for the first time has been a tried and true formula for establishing the academic significance of a piece of work, and helps with getting something published.
- We discussed briefly on the last slide what to do if a business or product owner asked for 99% accuracy and if you think that's unrealistic, measuring HLP may help you to establish a more reasonable target.
- Do be cautious about trying to prove that the Machine Learning System is superior to the human in doing a particular job. While it may be tempting to prove the superiority of your learning algorithm, as a practical matter, this approach rarely works.
- You also saw last week that businesses need systems that do more than just doing well on average test set accuracy.
- I urge you to use this type of logic with caution, or maybe even more preferably just don't use these arguments. I've usually found other arguments to be more effective than this when working with the business to see if they should adopt a Machine Learning System.

The problem with beating HLP as a "proof" of ML "superiority"

$nl \rightarrow$
 "Um... nearest gas station" $\leftarrow 70\%$ of labels
 "Um, nearest gas station" $\leftarrow 30\%$
 Two random labelers agree: $0.7^2 + 0.3^2 = 0.58$
 ML agrees with humans: $0.70 \leftarrow +12\%$

The 12% better performance is not important for anything! This can also mask more significant errors ML may be making.

- The problem with beating Human Level Performance as proof of machine learning superiority is multi fold. One of the problems with this metric is that it sometimes gives a learning algorithm an unfair advantage when labelling instructions are inconsistent.
- For example, we have an audio clip that says "nearest gas station". Let's say 70% of labellers uses label convention of "Um... nearest gas station" (using ellipsis) and 30% of labellers uses label convention of "Um, nearest gas station" (using comma).
- In fact both transcripts are completely fine, but just by luck of the draw, 70% of labellers tend to choose the first one and 30% choose the second one.
- As a result, the chance that two random labellers will agree will be $0.7^2 + 0.3^2$, which is 0.58.
- It means that given the first convention, there is a 0.7^2 chance that two labellers will both label the same way. If the second convention is used, there is a 0.3^2 chance that both random labellers will label it the same way.
- Overall, the chances of labellers agreeing is 0.58.
- And in the usual way of measuring Human Level Performance, you will conclude that Human Level Performance is 0.58.
- But actually what you're really measuring is really the chance of two random labellers agreeing.
- This is where the machine learning our room has an unfair advantage.
- Upon training, the learning algorithm may be able to always use the first labelling convention (with ellipsis) because it knows that statistically that it has a 70% chance of getting it right if it uses ellipses instead of comma.

- So a learning algorithm will agree with humans 70% of the time, just by choosing the first labelling convention.
- But this 12% improvement in performance (0.70 vs. 0.58) not actually that important other than selecting for a choice between two equally good, slightly arbitrary choices.
- The learning algorithm just consistently picks the first one so it gains what seems like a 12% advantage on this type of query, but it's not actually outperforming any human in any way that a user would care about.
- And one side effect of this is that, if the learning algorithm makes some more significant errors on other types of input audio, then the parts where it actually performs worse could be averaged out by queries like these
- And this will therefore mask or hide the fact that you're learning algorithm is actually creating worse transcripts than humans actually are, and creating a fake impression that the machine learning system is doing better than HLP.
- Overall, it could actually be producing worse transcripts than human transcription because it's just doing better on this type of problem, which is not important to do better on. It could potentially be actually doing worse on some other types of input audio.

Raising HLP

Raising HLP

When the ground truth label is externally defined, HLP gives an estimate for Bayes error / irreducible error. *e.g. biopsy*
 But often ground truth is just another human label.

Scratch length (mm)	Ground Truth Label	Inspector
0.7	1	1
0.2	1 0	0
0.5	1	1
0.2	0	0
0.1	0	0
0.1	0	1 0

0.3 mm (circled under 0.1 in Ground Truth Label column)
66.7% (handwritten)
 \downarrow
100% (handwritten)

- I've done a lot of work on medical imaging, working on AI for retrieving diagnosis from X-rays
- Given the task of predicting a diagnosis from a X-ray image, if the ground truth diagnosis is defined according to biological or medical tests (e.g. biopsy), then HLP helps you measure how well a doctor performs as compared to a learning algorithm
- But when the ground truth is defined by a human (e.g. X-ray image labelled by a doctor), then HLP is just measuring how well can one doctor predict another doctor's label
- That too is useful, but it's different than if you're measuring how well your algorithm compares with a doctor in predicting a ground truth outcome from a medical biopsy.
- To summarize, when the ground truth label is externally defined (e.g. medical biopsy), then HLP gives an estimate for base error and irreducible error in terms of predicting the outcome
- If the ground truth is just another human label (which is more often), then there will be problems
- Take for example the phone screen visual inspection example we had earlier.
- Rather than just aspiring to beat the human inspector, it may be more useful to see why the ground truth and the inspector don't agree.

- Looking at the lengths of the different scratches that they labelled (see table in slide above), if we speak of the inspectors and have them agree that 0.3 mm is the threshold above which a scratch becomes a defect, then what we realize is that for the first row, both labels of 1 are totally appropriate.
- For the second row, the ground truth here should now be 0 instead of 1 (since 0.2mm is less than the threshold of 0.3mm),
- If we go through this exercise of getting the ground truth label and this inspector to agree, then we actually just raised human-level performance from 66.7% to 100%, at least as measured on these six examples.
- But notice what we've done, by raising HLP to 100% we've made it pretty much impossible for learning algorithm to beat HLP,
- It might seem terrible that you can't tell the business owner anymore that your system can beat HLP, but the benefit of this is you now have much cleaner, more consistent data, and that ultimately will allow your learning algorithm to do better.
- The goal is to come up with a learning algorithm that actually generates accurate predictions rather than just prove that you can beat HLP

Raising HLP

- When the label y comes from a human label, $HLP \ll 100\%$ may indicate ambiguous labeling instructions. *Um, Um...*
 - Improving label consistency will raise HLP.
 - This makes it harder for ML to beat HLP. But the more consistent labels will raise ML performance, which is ultimately likely to benefit the actual application performance.
- To summarize, when the ground truth label y comes from a human, HLP being quite a bit less than 100% may just indicate that the labelling instructions or labelling convention is ambiguous.
 - We previously saw an example of this in visual inspection.
 - This type of ambiguous labelling convention will also cause HLP to be less than 100%.
 - Improving label consistency will then help to raise human-level performance, making it harder (unfortunately) for your learning algorithm to beat HLP
 - But overall, more consistent labelling will raise your machine learning algorithm performance, which is ultimately likely to benefit the actual application.

HLP on structured data

Structured data problems are less likely to involve human labelers, thus HLP is less frequently used.

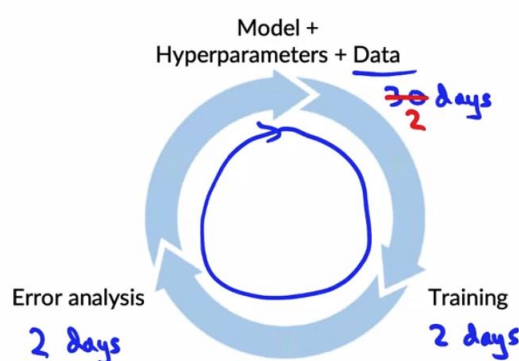
Some exceptions:

- User ID merging: Same person?
- Based on network traffic, is the computer hacked?
- Is the transaction fraudulent?
- Spam account? Bot?
- From GPS, what is the mode of transportation – on foot, bike, car, bus?

- So far we've been discussing HLP on unstructured data, but some of these issues apply to structured data as well.
- You already know that structured data problems are less likely to involve human labellers, and thus HLP is less frequently used.
- But there are exceptions. You saw previously the user ID emerging example, where you might have a human labeller to identify whether two records belong to the same person.
- I've worked on projects where we will look at network traffic into a computer to try to figure out if the computer was hacked, and we got human IT experts to provide labels for us.
- Sometimes it's hard to know if a transaction is fraudulent and you just ask a human to label that. Or is this a spam account or a bot-generated accounts?
- Or from GPS, what is the mode of transportation is this person on foot, or on a bike, or in the car, or the bus. It turns out buses stop at bus stops, so you can actually tell if someone is in a bus or in a car based on the GPS trace.
- For problems like these, it would be quite reasonable to ask a human to label the data, at least on the first pass for a learning algorithm to make such predictions.
- When the ground truth label you're trying to predict comes from one human, the same questions of what HLP means will apply.
- It is a useful baseline to figure out what is possible, but sometimes when measuring HLP, you realize that low HLP stems from inconsistent labels, and working to improve HLP by coming up with a more consistent labelling standard will both raise HLP and give you cleaner data with which to improve your learning algorithm performance.
- Here's what I hope you take away from this video. First, HLP is important for problems where human-level performance can provide a useful reference. I do measure it and use it as a reference for what might be possible and to drive analysis and prioritization.
- Having said that, when you're measuring HLP, if you find the HLP is much less than 100%, also ask yourself if some of the gaps in HLP is due to inconsistent labelling instructions.
- Because if that turns out to be the case, then improving labelling consistency will raise
- HLP and also give cleaner data for your learning algorithm, which will ultimately result in better machine-learning algorithm performance.

Obtaining data

How long should you spend obtaining data?



- Get into this iteration loop as quickly possible.
- Instead of asking: How long it would take to obtain m examples?
Ask: How much data can we obtain in k days.
- Exception: If you have worked on the problem before and from experience you know you need m examples.

- One key question to think about is how much time should you spend obtaining data?
- You know that machine learning is a highly iterative process where you need to pick a model, hyperparameters, have a data set, perform training, carry out error analysis, and then go around this loop multiple times to get to a good model.

- Let's say training your model for the first time takes a couple of days, and carrying out error analysis for the first time may take a couple of days.
- If this is the case, I would urge you not to spend **30** days collecting data, because that will delay by a whole month before getting into this iteration.
- Instead, I urge you to get in this iteration loop as quickly as possible.
- I would urge you to ask yourself, what if you were to give yourself only two days to collect data. Would that help get you into this loop much more quickly?
- Maybe two days is too short, but I've seen far too many teams take too long to collect their initial data set before they train the initial model.
- After you've trained your initial model, and carried out error analysis, there's plenty of time to go back and collect more data.
- I found for a lot of projects I've led when I go to the team and say, "Hey everyone, we're going to spend at most seven days collecting data, so what can we do?" I found that posing the question that way often leads to much more creative ways of getting data
- That allows you to enter this iteration loop more quickly and let the project progress faster
- One exception to this guideline is if you have worked on this problem before, and from experience you know you need at least a certain training set size. If so, then it might be okay to invest more effort up front to collect that much data.
- Because I've worked on speech recognition, I have a good sense of how much data I'll need to do certain things and I know it's just not worth trying to train certain models if I have less than certain number of hours of data.
- But a lot of the time, if you're working on a brand new problem and if you are not sure (and it is often hard to tell even from the literature), it is much better to quickly collect a small amount of data, train a model and use error analysis to tell you if it is worthwhile to collect more data.

Inventory data

Brainstorm list of data sources ( speech recognition)





Source	Amount	Cost	Time	
Owned	100h	\$0	0	✓
Crowdsourced – Reading	1000h	\$10000	14 days	
Pay for labels	100h	\$6000	7 days	
Purchase data	1000h	\$10000	1 day	✓

Other factors: Data quality, privacy, regulatory constraints

- In terms of getting the data, one step I often carry out is to take inventory of possible data sources.
- Let's continue to use speech recognition as an example. If you were to brainstorm a list of data sources, this is maybe what you might come up with.
- Maybe I already own 100 hours of transcribed speech data, and because you already own it, the cost of that is zero.
- Or you may be able to use a crowdsourcing platform and pay people to read text. You provide them a piece of text and ask them to read it out loud. This creates the text data you need since you already have the transcript and they were reading off it
- Or you may decide to take audio that you have that hasn't been labelled yet, and to pay for it to be transcribed. It turns out this is more expensive on a per hour basis than paying people to read texts, but this results in audio that sounds more natural because people aren't reading it

- Or you may find some commercial organizations that could sell you data.
- Through an exercise like this, you can brainstorm what are the different types of data you might use, as well as their associated costs.
- One column that's missing from this, that I find very important, is the time costs i.e. how long will it take you to get these different types of data?
- For the owned data you could get that instantaneously. For crowdsourced reading, you may need to implement a bunch of software, find the right crowdsourcing platform, and carry out software integration, so you might estimate that that's two weeks of engineering work.
- Paying for data to be labelled is simpler but work is still needed to organize and manage, whereas purchasing data may be much quicker.
- I find it some teams won't go through an inventory process like this, and would just pick a random idea. But if you can sit down, write down all the different data sources and think through the trade-offs, including costs and time, then that can help you to make better decisions
- If you are especially pressed with time, based on this analysis, you may decide
- to use the data you already own and maybe purchase some of it
- In addition to the amount of data you can acquire, and the financial costs and the time costs, other important factors that's application dependent will include data quality, privacy and regulatory constraints.

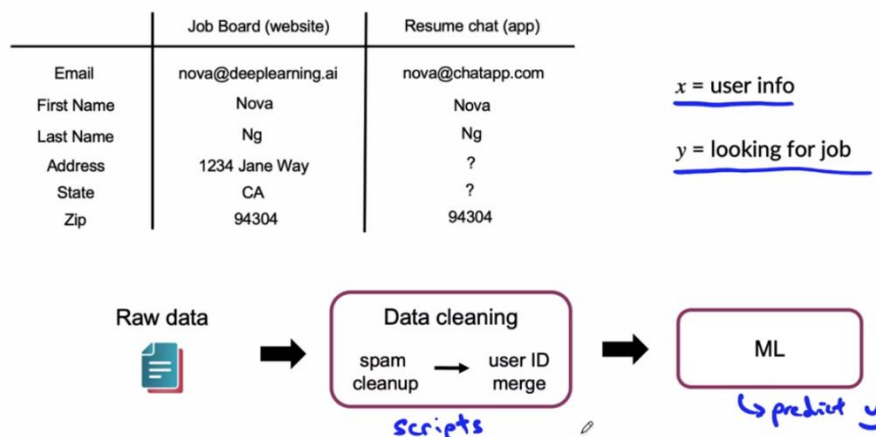
Labeling data

- Options: In-house vs. outsourced vs. crowdsourced
 - Having MLEs label data is expensive. But doing this for just a few days is usually fine.
 - Who is qualified to label? 
 -  Speech recognition – any reasonably fluent speaker
 -  Factory inspection, medical image diagnosis – SME (subject matter expert)
 -  Recommender systems – maybe impossible to label well
 - Don't increase data by more than 10x at a time
-
- The three most common ways to get data labelled are: **in-house**, where you have your own team label the data, versus **outsource**, where you might find some company that labels data and have them do it for you, versus **crowdsourced**, where you might use a crowdsourcing platform to have a large group collectively label the data.
 - The difference between outsource versus crowdsourced is that, depending on what type of data you have, there may be specialized companies that could help you get the label quite efficiently.
 - Having machine learning engineers label data is often expensive, but I find that to get a project going quickly, having machine learning engineers do labelling just for a few days is usually fine
 - In fact, this can help build the machine learning engineers' intuition about the data.
 - When I'm working on a new project, **I often don't mind spending a few hours or maybe a day or two labelling data myself, if that helps me to build my intuition about the project.**
 - But beyond a certain point you may not want to spend all your time as a machine learning engineer labelling data, and you might want to shift to a more scalable labelling process.
 - Depending on your application, there may be also different groups or subgroups of individuals that are going to be more qualified to provide the labels
 - If you're working on speech recognition, then maybe almost any reasonably fluent speaker can listen to audio and transcribe it.
 - For more specialized applications like factory inspection or medical image diagnosis, a typical person off the street probably can't look at a medical X-ray image and diagnose from it, or look at a smartphone and determine what a defect is. More specialized tasks like these usually require an SME or subject matter expert, in order to provide accurate labels.

- Finally there are some applications that are very difficult to get anyone to give good labels. Take product recommendations, there are probably product recommendation systems that are giving better recommendations to you than even your best friends or maybe your significant other.
- For this, you may just have to rely on purchase data by the user as a label rather than get humans to label this.
- When you're working on an application, figuring out which of these categories of application you're working on and identifying the right type of person to help you label, will be an important step to making sure your labels are high quality.
- One last tip. Let's say you have 1,000 examples, and you've decided you need a bigger data set. How much bigger should you make your data set?
- One tip I've given a lot of teams is **don't increase your data by more than 10x at a time.**
- If you have trained your model on 1,000 examples, maybe it's worth investing to try to increase your dataset to 3,000 examples or maybe at most 10,000 examples.
- But I would first do a less than 10x increase, train another model, carry out error analysis and only then figure out if it's worth increasing it substantially beyond that.
- Because once you increase your dataset size by 10x, so many things change, and it becomes really hard to predict what will happen when your data set size increases even beyond that.
- This guideline hopefully will help teams avoid over investing in tons of data, only to realize that collecting quite that much data wasn't the most useful thing

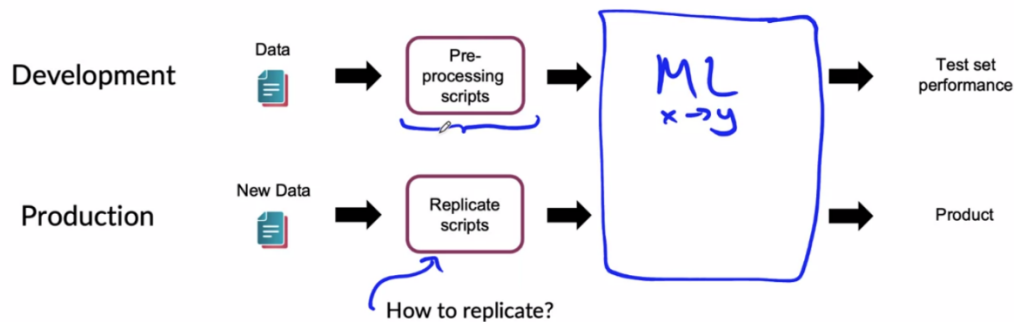
Data pipeline

Data pipeline example



- Let's say that given some user information, you would like to predict if a given user is looking for a job, and then surface job ads or other useful information to them.
- Given raw data such as the data on top, there's often some pre-processing or data cleaning before the data is fed to the learning algorithm
- The data cleaning may include things like spam clean-up, such as removing the spam accounts, and maybe also user ID merge, which we talked about in an earlier video.
- Let's say these data cleaning is done with just scripting (explicit sequences of instructions). These could be done with machine learning algorithms as well, but it makes them a bit more complex to manage.

Data pipeline example



- When you have scripts for the data cleaning, one of the issues you run into is replicability when you take these systems into production deployment.
- During development phase, you may have seen that pre-processing scripts can be quite messy. It may be you hacking something up, processing data, mailing a file to a different member of your team, having them have a few incantations in Python etc.
- The key question is, if your pre-processing was done with a bunch of scripts, spread out on a bunch of different people's computers, how do you replicate the scripts to make sure that the input distribution to a machine learning algorithm was the same for the development data and the production data?
- I find that the amount of effort that you should invest to make sure that the pre-processing scripts are highly replicable depends on the phase of the project.

POC and Production phases

POC (proof-of-concept):

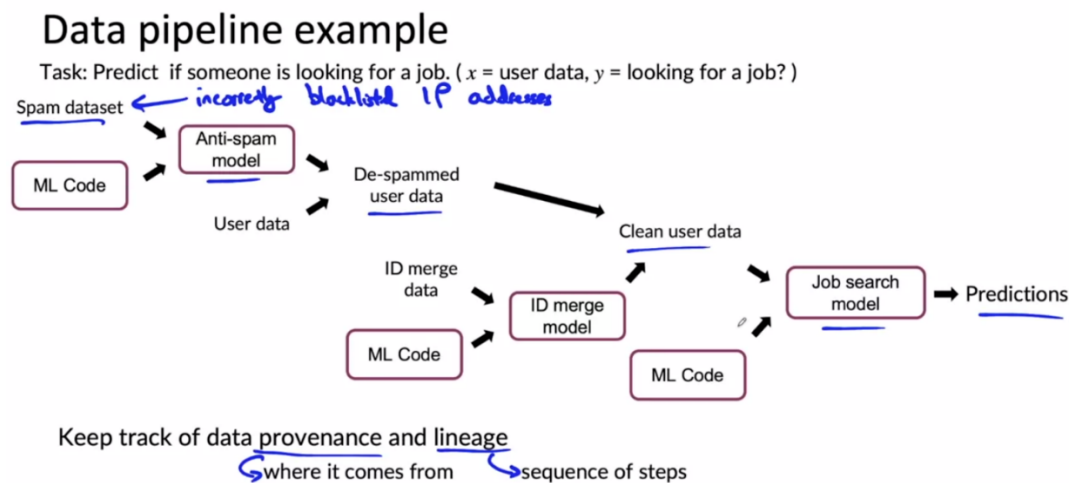
- Goal is to decide if the application is workable and worth deploying.
- Focus on getting the prototype to work!
- It's ok if data pre-processing is manual. But take extensive notes/comments.

Production phase:

- After project utility is established, use more sophisticated tools to make sure the data pipeline is replicable.
- E.g., TensorFlow Transform, Apache Beam, Airflow,....

- A lot of projects go through a proof of concept or POC phase, and then a production phase
- During the proof of concept phase, the primary goal is just to decide if the application is workable and worth building and deploying.
- My advice to most teams is during the POC phase, focus on getting the prototype to work.
- It's okay if some of the data pre-processing is manual. If the project succeeds, you can then replicate all of this pre-processing later.
- My advice would be take extensive notes, write extensive comments to increase the odds that you can replicate all this pre-processing later, but this is also not the time to get bogged down in tons of processes just to ensure replicability
- The focus is really to decide if the application is workable and is worth taking to the next phase.
- Once you decided that this project is worth taking to production, then you know it's going to be really important to do the replication of the pre-processing scripts.
- In this phase, that's when I would use more sophisticated tools to make sure the entire data pipeline is replicable. Tools like TensorFlow Transform, Apache Beam, and Airflow thus become very valuable.

Meta-data, data provenance and lineage



- Here's a more complex example of a data pipeline. Let's say you start off with a spam dataset. This may include a list of known spam accounts as well as features of blacklisted IP addresses
- You then implement an algorithm for spam detection.
- You then take your user data and apply the anti-spam model to get de-spammed user data.
- Now, taking your de-spammed user data, you may want to carry out user ID merge.
- To do that, you might start off with some ID merged data. This would be labelled data telling you some pairs of accounts that actually correspond to the same person
- We then have a machine learning algorithm implementation, train the model on that, and this gives you a learned ID merge model
- We apply it to the de-spammed user data, and that gives us the cleaned up user data.
- Finally, based on the clean user data, you'll then have another machine learning model to predict if a given user is looking for a job or not.
- I've seen data pipelines or data cascades that are even far more complicated than this.
- One of the challenges of working with data pipelines like this is, what if after running this system for months, you discover that the IP address blacklists you're using has some mistakes in it. (e.g. incorrectly blacklisted IP addresses because multiple users use it, like in a university campus)
- The question is, having built up this big complex system, if you were to update your spam dataset, and all the different parts in the pipeline? How do you go back and fix this problem?
- This is especially challenging if each of these systems was developed by different engineers, and you have files spread across the laptops of your machine learning engineering development team.
- To make sure your system is maintainable, it can be very helpful to keep track of data provenance as well as lineage.
- Data provenance refers to where the data came from e.g. from whom did you purchase the spam IP address from?
- Lineage refers to the sequence of steps needed to get to the end of the pipeline.
- At the very least, having extensive documentation could help you reconstruct data provenance and lineage, helping to build robust, maintainable systems in the production stage.
- To be honest, the tools for keeping track of data provenance and lineage are still immature in this machine learning world. I find that extensive documentation can help and some formal tools like TensorFlow Transform can also help, but solving this type of problem is still not something that we are great at as a community yet.

Meta-data

Examples:



Manufacturing visual inspection: Time, factory, line #, camera settings, phone model, inspector ID,....



Speech recognition: Device type, labeler ID, VAD model ID,....

Useful for:

- Error analysis. Spotting unexpected effects.
- Keeping track of data provenance.

- Metadata is data about data.
- For example, in manufacturing visual inspection, the data would be pictures of phones and its labels, and the metadata tells you what time was the picture taken, what factory was this picture from, what's the line number, what were the camera settings such as camera exposure time and camera aperture, what's the ID of the inspector that provided this label etc.
- This type of metadata can turn out to be really useful because if you discover during machine learning development, that for some strange reason, some samples are producing a lot more errors, This allows you to go back to the source to investigate it during error analysis
- I found many times when I store the right metadata, that metadata helped generate a key insight that helped the project move forward.
- My tip is if you have a framework or a set of MLOps tools for storing metadata, it will definitely make life easier, just like you rarely regret commenting your code
- In the same way, if you don't store the metadata in a timely way, it can be much harder to go back to recapture and organize that data.
- One more example for speech recognition.
- If you have audio recordings from different brands of smartphones, the voice activity detection model used can vary. If there is something wrong with any of these VAD models, having metadata significantly increases the odds of you discovering the errors and use it to improve your algorithm's performance.
- To summarize, metadata can be very useful for error analysis and spotting unexpected effects, categories of data that have some unusually poor performance
- As part of building out the systems, consider keeping track of metadata, which can help you with tracking data provenance but also error analysis.

Balanced train/dev/test splits

Balanced train/dev/test splits in small data problems



Visual inspection example: 100 examples, 30 positive (defective)

Train/dev/test: 60%/20%/20%

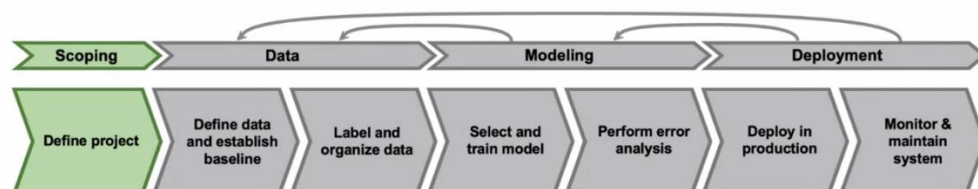
Random split: 21/2/7 positive example
35% 10% 35%

Want: 18/6/6 } balanced split
30%/30%/30%

No need to worry about this with large datasets – a random split will be representative.

- Many of us are used to taking a data set and randomly splitting it into train, dev and test sets
- It turns out when your data set is small, having balanced train, dev, and test set can significantly improve your machine learning development process.
- Let's use our manufacturing visual inspection example. Say your training set has 100 images (pretty small data set) and with 30 positive examples, so 30 defective phones and 70 non defective.
- If you were to use a split of 60% of the data in the training set, 20% in the dev (or validation) set and 20% in the test set, then by chance in the random split you may end up with 21 positive examples in train, 2 in dev and 7 in test.
- This would be quite likely just by random chance.
- And this means that the training set is 35% positive, not that far from 30% positive in the overall dataset, but your dev set is only 10% positive and your test set is 35% positive.
- And this makes your dev set quite non representative, because it has only 2 (or 10%) positive examples rather than 30% positive examples.
- So what we would really want is for the training set to have exactly 18 positive examples, dev set to have exactly 6 positive examples and the test set to have exactly 6 positive examples. And this would be 30%, 30% 30%.
- And if you could get this type of split, this would be called a **balanced split**, where each of your train, dev and tests has exactly 30% positive examples, and this makes your data set more representative of the true data distribution.
- There's no need to worry about this effect when you have a large data set, as a random split will very likely be representative, meaning that the percentage of positive examples will be quite close to your overall data set.
- This is one of those little techniques that turns out to make a big difference to your performance when you're working on a small data problem

What is scoping



Scoping example:  Ecommerce retailer looking to increase sales 

- Better recommender system
- Better search
- Improve catalog data
- Inventory management
- Price optimization

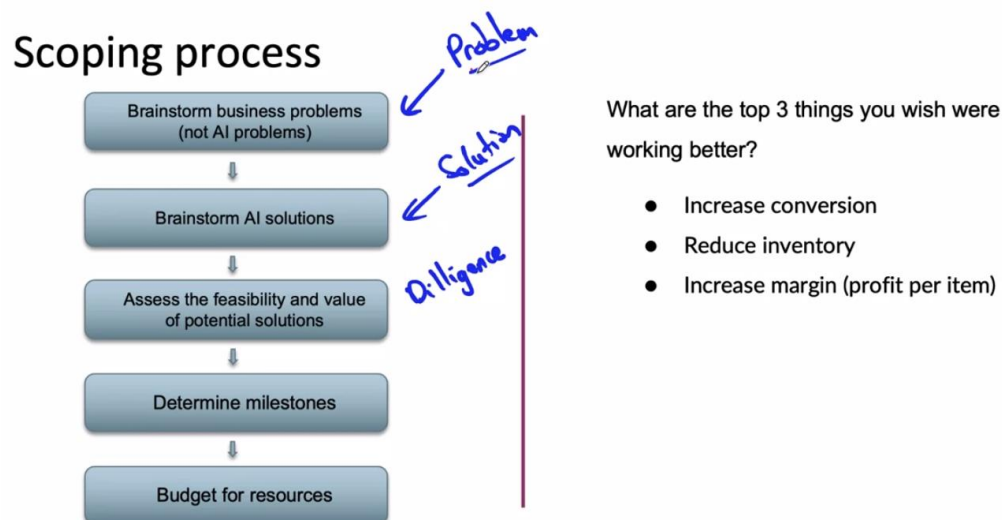
Questions:

- What project should we work on?
- What are the metrics for success?
- What are the resources (data, time, people) needed?

- Picking the right project to work on is one of the most rare and valuable skills in AI today.
- I remember when I was younger, I tended to just jump into the first project that I got excited about and sometimes I was lucky and it worked out okay.
- Now that I've had a little bit more experience, I find that it is well worth your while to spend plenty of time to think through a few options and select the most promising project to work on before putting so much effort into it. So let's dive into scoping.
- Let's use the example of an e commerce retailer looking to increase sales. If you were to sit down and brainstorm, you might come up with many ideas, such as maybe a better product recommendation system, or a better search so people can find what they're looking for, or you may want to improve quality of the catalogue data

- You may even help them with inventory management, such as deciding how many shirts to buy, where to ship the shirts or what price optimization.
- With a quick brainstorming session, you may be able to come up with dozens of ideas for how to help this e commerce retailer.
- The questions that we like to answer with scoping, is what project or projects should we work on.
- We also want to find out what the metrics for success are, and what are the resources needed to execute this project.
- What I've seen in a lot of businesses is that of all the ideas you can work on, some are going to be much more valuable than others.
- Being able to pick the most valuable project will significantly increase the impact of your work

Scoping process



- The first thing I do is usually get together with a business or private owner (someone that understands the business) and brainstorm with them. What are their business or application problems?
- And at this stage I'm trying to identify a business problem, not an AI problem.
- I might ask for the top three things you wish were working better. This could be business problems like increasing conversions, reducing inventory, increasing profit per item sold etc.
- At this point in the process, I'm not trying to identify an AI problem. In fact, I often tell my partners, I don't want to hear about your AI problems. I want to hear about your business problems.
- It's my job to work with you to see if there is an AI solution. Sometimes there isn't and that's fine. Feel free to use the same words when brainstorming projects with your non-AI partners.
- Once you have identified a few business problems, only then do I start to brainstorm if there are possible AI solutions. Not all problems can be solved by AI and that's okay.
- It is helpful to separate out the identification of the problem, from the identification of the solution
- As engineers, we are pretty good at coming up with solutions, but having a clear articulation of what is the problem first often helps us come up with better solutions.
- After brainstorming a variety of different solutions, I would then assess the feasibility and the value of these different solutions.
- Sometimes you hear me use the word diligence. Diligence is a term that actually comes from the legal field, but it basically means double-checking if an AI solution really is technically feasible and valuable
- If it still looks promising, we then flesh out the milestones for the project and then budget for resources

Separating problem identification from solution

Problem	Solution
Increase conversion	Search, recommendations
Reduce inventory	Demand prediction, marketing
Increase margin (profit per item)	Optimizing what to sell (e.g., merchandising), recommend bundles

- Let's take a deeper look at this process of identifying problems and solutions
- So the first one is increased conversion
- You may have different ideas as to how to do that. For example, you may want to improve the quality of the website search results, so people find more relevant products when they search. Or you might decide to try to offer a better product recommendations based on their purchase history.
- It is quite common that one problem may lead to multiple ideas for solutions.
- You may be able to bring some other ideas as well, such as maybe a redesign of how products are displayed on the page.
- Or you may find interesting ways to surface the most relevant product reviews, to help users understand the product and hopefully purchase it.
- Take the next problem of reducing inventory. You can imagine doing up a demand prediction project to better estimate how many people will buy something from you.
- This will then help you keep a more accurate inventory in your warehouses.
- Or you may decide to come up with a marketing campaign to drive sales for the products that you bought too many of, so as to reduce inventory
- There could be numerous ideas for solutions for the problem of increasing margin.
- You may come up with some ways to use machine learning to optimize what to sell.
- In retail, sometimes this is called merchandising, just deciding what to sell.
- You can recommend bundles where if someone buys a camera, or you can recommend to them a protective camera case so as to increase margin.
- The problem identification is a step of thinking through what are the things you want to achieve.
- The solution identification is a process of thinking through how to achieve those objectives.
- One thing I see too many teams do today is jumping into the first project that they're excited about
- I find it worthwhile to first engage in divergent thinking, where you brainstorm a lot of possibilities. Next, follow it up with convergent thinking where you then narrow it down to one or a small handful of the most promising projects to focus on.

Feasibility: Is this project technically feasible?

Use external benchmark (literature, other company, competitor)

	Unstructured (e.g., speech, images)	Structured (e.g., transaction records)
New	HLP	Predictive features available?
Existing	HLP History of project	New predictive features? History of project

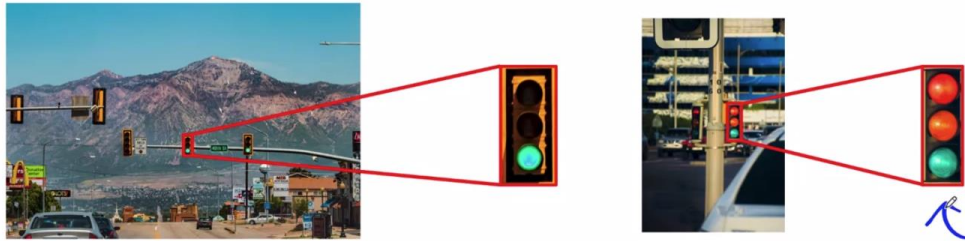
HLP: Can a human, given the same data, perform the task?

- Let's start with finding out whether the project idea is technically feasible
- One way to get a quick sense of feasibility is to use an external benchmark, such as existing research, literature or other forms of publications
- This can even be information from other competitors or other companies
- That might help give you a sense that this project may be technically feasible, because someone else has managed to do something similar.
- Here are some other ways to assess feasibility as well, using a two by two matrix. On one axis, we see if our problem has unstructured data like images, or structured data like transaction records.
- On the other axis, I'm going to put new versus existing. New means you are delivering a brand new capability, while existing means you're scoping out the project to improve on an existing capability.
- In the upper left hand quadrant (new unstructured), HLP to be very useful in giving you an initial sense of whether a project is doable.
- When evaluating HLP, I would give a human to assess the same data as would be fed to a learning algorithm
- For example, given images of scratched smartphones, can a human perform the task of detecting scratches reliably?
- If a human can do it, then that increases the hope that we can also get a learning algorithm to do it.
- For existing projects, I would use HLP as a reference as well. If humans can achieve the level you're hoping to get to, then that might give you more hope that it is technically feasible.
- Whereas if you're hoping to increase performance well beyond human level performance, then it suggests that the project might be harder, or may not be possible.
- In addition to HLP, I often also use the history of the project (e.g. previous rate of progress) as a predictor for future progress.
- Moving over to the right column. If you're working on a brand new project with structured data, the question I would ask is whether predictive features are available
- Do you have reason to think that the data you have (inputs X) are strongly predictive or sufficiently predictive of the target outputs Y?
- In this box on the lower right quadrant, if you're trying to improve an existing system with structured data, one thing that will help a lot is if you can identify new predictive features.
- Are there features that you aren't yet using but could really help predict Y
- We will also look at the history of the project

Why use HLP to benchmark?






People are very good on unstructured data tasks

Criteria: Can a human, given the same data, perform the task?



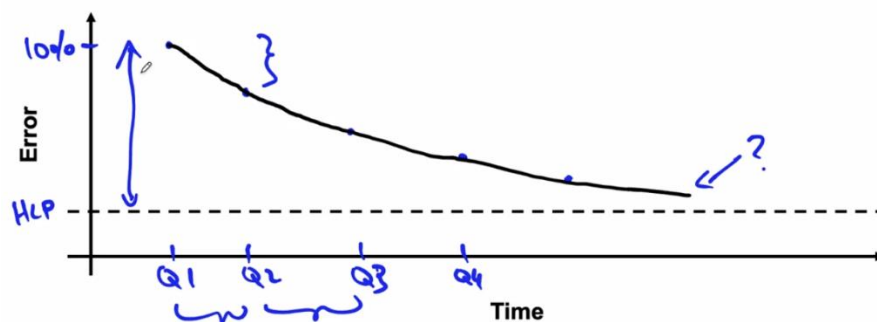
- I use HLP to benchmark what might be feasible for unstructured data because people are very good on unstructured data tasks.
- The key criteria for assessing project feasibility is, given the exact same data as would be given to a learning algorithm, can the human perform the task?
- Let's say you're building a self-driving car, and you want an algorithm to classify whether a traffic light is currently red, yellow, or green.
- I will take pictures from the self-driving car and ask a person to look at an image like this (left of the slide), and see if the person looking only at the image can tell which lamp is illuminated. In this example, it's pretty clear it's green.
- But if you find that you also have pictures like these (right of slide), where we can't tell which lamp is illuminated, it will become an issue.
- This is why it's important for this HLP benchmark to make sure that human is given only the same data as your learning algorithm.
- The useful test is not whether the human eye can recognize which lamp is illuminated while in the car. The useful test is if the person can still do the task if he was sitting in the office and they can only see the image from the camera
- Specifically, it helps you make a better guess at whether a learning algorithm, which will only have access to this image, can also accurately detect which lamp in the traffic light is illuminated.
- Making sure that a human sees only the same data as a learning algorithm is really important.
- I've seen a lot of projects where for a long time a team was working on a computer vision system, only to find out even a human looking at the image couldn't figure out what was going on.
- If you can realize that earlier, then you can figure earlier that it was just not feasible with the current camera setup
- The more efficient thing to do would have been to invest early on in a better camera or better lighting setup, rather than keep working on the machine learning algorithm

Do we have features that are predictive?

-  Given past purchases, predict future purchases ✓
-  Given weather, predict shopping mall foot traffic ✓
-  Given DNA info, predict heart disease ?
-  Given social media chatter, predict demand for a clothing style ?
-  Given history of a stock's price, predict future price of that stock ✗

- For structured data problems, one of the key criteria to assess for technical feasibility is whether we have input features X that seem to be predictive of output Y.
- In e-commerce, if you have features that show what the past purchases of a user are, it is possible to predict future purchases, because previous purchases are predictive of future purchases.
- If you work with a physical store and you want to predict shopping mall foot traffic, one predictive feature can be the weather. We know that when it rains a lot, fewer people leave their house, so weather is predictive of foot traffic in shopping malls
- Let's look at some more examples. Given DNA of an individual, let's try to predict if this individual will have heart disease. This one, I don't know, because the mapping from genotype to phenotype, or your genetics to your health condition, is a very noisy mapping.
- I would have mixed feelings about this project, because it turns out your genetic sequence is only slightly predictive of whether you get heart disease.
- In another example, given social media chatter, can you predict demand for a clothing style?
- This is another iffy one. I think you may be able to predict demand for clothing style right now, can you predict what will be the fashionable trend six months from now? That actually seems very difficult.
- Sometimes the data just is not that predictive, and you end up with a learning algorithm that does barely better than random guessing.
- That's why looking at whether you have features that you believe are predictive, is an important step of diligence for assessing the technical feasibility of a project.
- One last example that may be even clearer, is given a history of a particular stock market shares price, can you predict the future price of that stock.
- All of the evidence I've seen is that this is not doable or feasible, unless you get some other clever set of features looking at a single shares historical price.
- Past history is not predictive of the future price of that stock based on the evidence I've seen.
- Even leaving aside the question of whether such stock predictions have any social value, I also think this project is just not technically feasible

History of project



- One last criteria I mentioned is the history of a project.
- When I've worked on a machine learning application for many months, I found that the rates of previous improvements can be a surprisingly good predictor for the rate of future improvement.
- Let's see a speech recognition as the example, and let's say that this is human level performance.
- I'm going to use HLP as our estimate for Bayes or irreducible level of error that we hope to get to.
- Let's say that when you started a project, you'll see in the first quarter that the system had 10% error rate. Over time, in subsequent quarters, the error went down further across Q2, Q3, and Q4.
- It turns out that it's not a terrible model to estimate this curve.
- If you want to estimate how well the team could do in the future, one simple model I've used is to estimate the rate of error rate reduction for every fixed period of time (e.g. every quarter) relative to human level performance.
- In this case, it looks like this gap between the current level of performance and human level performance is shrinking by maybe 30% every quarter, which is why you get this curve that is exponentially decaying to what HLP is
- By estimating this rate of progress, it will give you a sense of what might be reasonable for the future rate of progress on this project.

Diligence on value

Diligence on value



Have technical and business teams try to agree on metrics that both are comfortable with.

- How do you estimate the value of Machine Learning Project? This is sometimes not easy to estimate, but let me share with you a few best practices.
- Let's say you're working on building a more accurate speech recognition system for the purpose of voice search
- It turns out that in most businesses, there will be some metrics that machine learning engineers are used to optimizing, and some metrics that the owners of the product or business will want to maximize. There's often a gap between these two.
- But many machine learning teams would be comfortable optimizing for good word-level accuracy.
- But in a business context, one other key metric is query-level accuracy, which is how often do you get all the words in a query correct.
- For some businesses, word-level accuracy is important, but query-level accuracy may be even more important.
- We've now taken a step away from the objective that the learning algorithm is directly optimizing.
- Even after you get the query right, the thing that users care even more about is the search result quality. The reason the business may want to ensure search result quality, is that it gives users a better experience and thereby increasing user engagement. This in turn will make them come back to the search engine more often.
- One gap I've often seen between machine learning teams and business teams is the engineering team will usually want to work on this (i.e. word level accuracy), whereas the business leader may want to work on this (i.e. revenue)

- In order for a project to move forward, I usually try to have the technical and business teams agree on metrics that both are comfortable with.
- This often takes a little bit of compromise, where the machine learning team might stretch a little bit further to the right and the business teams stretch a little bit further to the left.
- **The further we go to the right, the harder it is for a machine learning team to really give a guarantee in terms of results**
- A lot of practical problems require that we do something more than just optimizing test accuracy.
- Having the technical team and the business teams both step a little bit outside their comfort zone is often important for reaching compromise on a set of metrics that the technical team can deliver and business team feels can create sufficient value for the business
- One other practice I've found useful is to do up rough calculations to relate the level of accuracy with the metrics.
- For example, if word accuracy improves by one percent, based on rough guess, it can be assumed that it will consequentially improve query level accuracy by maybe 0.7 percent or 0.8 percent.
- This can then be continued to estimate how much this will then improve search result quality and user engagement and ultimately revenue,
- You can create these kind of crude calculations using concepts like Fermi estimates.
- Such back of envelope calculations can be a way to help bridge the machine learning team metrics and business metrics.

Ethical considerations

- Is this project creating net positive societal value?
 - Is this project reasonably fair and free from bias?
 - Have any ethical concerns been openly aired and debated?
- As part of estimating the value of a project, I would encourage you to give thought to ethical considerations as well, such as whether this project is creating net positive societal value. If not, I hope you won't do it.
 - I also encourage you to think through whether this project is reasonably fair and free from bias
 - Issues of values and ethics are very domain-dependent, and it can be very different in making loans versus healthcare versus recommending products online.
 - I encourage you to look up any ethical frameworks that have been developed for your industry and your application.
 - Ultimately, if you don't think the project you're working on will help other people or will help humanity move forward, I hope you'll keep searching for other, more meaningful projects to jump into.
 - In my work, I have faced difficult choices where I really wasn't sure if a particular project was something I should work on.
 - I found that having a team debate and discussing it openly often helps us get to better answer and feel more comfortable with whatever decision we make.
 - I have called off multiple projects even though the project was economically sound, because I didn't think it would help people

Milestones & Resourcing

Key specifications:

- ML metrics (accuracy, precision/recall, etc.)
- Software metrics (latency, throughput, etc. given compute resources)
- Business metrics (revenue, etc.)
- Resources needed (data, personnel, help from other teams)
- Timeline

If unsure, consider benchmarking to other projects, or building a POC (Proof of Concept) first.

- Determining milestones and resourcing involves writing out the key specifications for your project.
- This will include machine learning metrics such as accuracy or precision-recall. For some applications, this may also include fairness types of metrics.
- The specifications will often also include software metrics regarding the software system such as latency, throughput, queries per second, given computational resources available
- You might also write out estimates of the business metrics you hope to move for the project you're scoping, such as how much incremental revenue
- In addition, write out the resources needed, such as how much data? From which teams? Involve which personnel or need any help from cross-functional teams?
- Finally, the timeline on which you hope to achieve certain milestones or deliverables.
- If you find that you're having a very hard time writing on some of these key specifications, then you might also consider carrying out a bench-marking exercise to compare it to other similar projects that others may have worked on before, or building a proof of concept first in order to get a better sense of what metrics might be feasible,
- Only after you've done that POC can you then use that information to more confidently scope out the milestones and resources needed for a larger scale execution of the project you have in mind.

References

- [Label ambiguity](#)
- [Data pipelines](#)
- [Data lineage](#)
- [MLops](#)
- Geirhos, R., Janssen, D. H. J., Schutt, H. H., Rauber, J., Bethge, M., & Wichmann, F. A. (n.d.). Comparing deep neural networks against humans: object recognition when the signal gets weaker*. Retrieved May 7, 2021, from Arxiv.org website: <https://arxiv.org/pdf/1706.06969.pdf>