Course 1 – 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
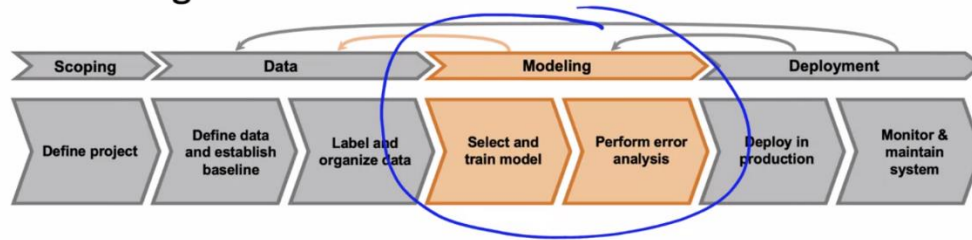
Week 2: Select and Train a Model

Contents

Modelling Overview

Modeling

| Scoping | Data | Modeling | Deployment |

- One of the themes you hear me refer to multiple times is model-centric AI development versus data-centric AI development.
- The way that AI has developed, there's been a lot of emphasis on how to choose the right model, such as how to choose the right neural network architecture.
- I found that for practical projects, it can be even more useful to take a more **data-centric** approach, where you focus not just on improving the neural network architecture, but on making sure you are feeding your algorithm with high-quality data.
- That lets you be more efficient in getting your system to perform well. But the way I engage in data-centric AI development is not to just go and collect more data, which can be very time-consuming, but to instead use tools to help me improve the data in the most efficient way possible.

Key Challenges

AI system = Code + Data
(algorithm/model)

- One framework that I hope you keep in mind when developing machine learning systems is that, AI systems of machine learning systems comprise both code, meaning the algorithm or the model, as well as data.
- There's been a lot of emphasis in the last several decades on how to improve the code.
- In fact a lot of research involve researchers downloading datasets and trying to find an overall model that does well on the dataset.
- But for many applications, you have the flexibility to change the data if you don't like the data. And so, there are many projects where the algorithm or model is basically a solved problem.
- Some model you download off GitHub will do well enough, and it will be more efficient to spend a lot of your time improving the data, because the data is much more customized to your problem.

Model development is an iterative process
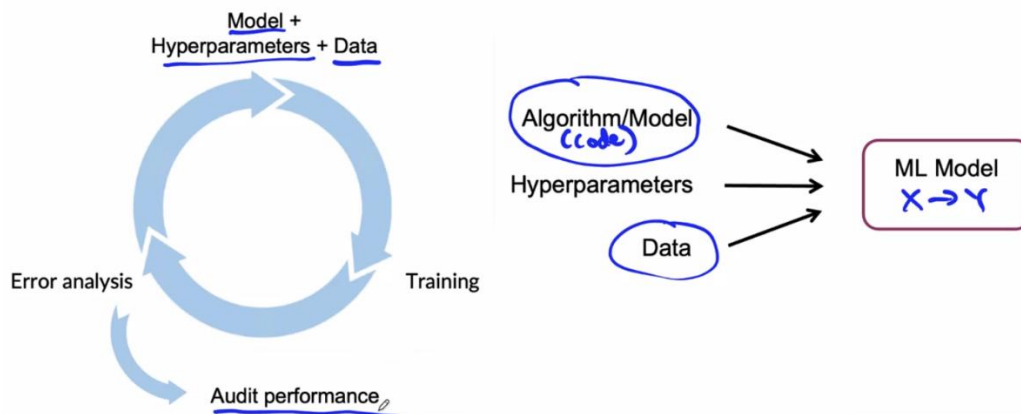
- Diving into more detail, when building a machine learning system, you may have an algorithm or a model, this would be your code and some data.
- Of course hyperparameters are an additional input to this process. It is important for many applications to make sure you have tuned learning rates and regularization parameters etc.
- Because the space of hyperparameters is usually relatively limited, so I'm going to spend more of our time focusing on the data and on the code.
- Model development is a highly iterative process. Because machine learning is such an empirical process, being able to go through this loop many times quickly is key to improving performance.
- But one of the things that will help you improve performance, each time through the loop, is being able to make good choices about how to modify the data or how to modify the model or how to modify the hyper parameters.
- After you've done this enough times and achieved a good model, one last step that's often useful is to carry out a richer error analysis and have your system go through a final audit to make sure that it is working before you push it to a production deployment.

Challenges in model development

1. Doing well on training set (usually measured by average training error).

2. Doing well on dev/test sets.

3. Doing well on business metrics/project goals.
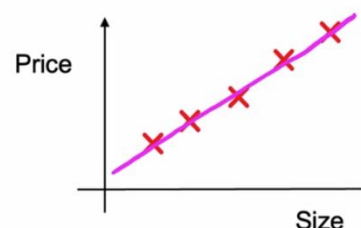
- When building a model, there are **three key milestones** that projects should aspire to accomplish.
- First you want to make sure you do well on the training set. So, if you're predicting housing prices as a function of the size of a house, are you at least able to fit a line in your training set quite well?
- After you've done well on the training set, you then have to ask if your algorithm does well on the development set or the holdout cross validation set, and then finally the test set.

- If your algorithm isn't even doing well on the training set, then it's very unlikely to do well on the dev set or the test set.
- After you do well on the dev set or test set, you also have to make sure that your learning algorithm does well according to the business metrics or according to the project's goals.
- Over the last several decades, a lot of machine learning development was driven by the goal of doing well on the dev set or test set.
- Unfortunately for many problems, having a high tested accuracy is not sufficient for achieving the goals of the project. And this has led to a lot of frustration and disagreements between the machine running team, which is very good at doing this and business teams which care more about the business metrics or some other goals of the project.

Why low average error isn't good enough

Performance on disproportionately important examples

Web Search example

"Apple pie recipe"   "Latest movies"   } Informational and Transactional queries

"Wireless data plan"   "Diwali festival"

"Stanford"   "Reddit"   "Youtube"   } Navigational queries

- A machine learning system may have low average test set error, but if its performance on a set of disproportionately important examples isn't good enough, then the machine learning system will still not be acceptable for production deployment.
- Let me use an example from Web search. There are a lot of web search queries like these: Apple pie recipe, latest movies, and wireless data plan.
- These types of queries are sometimes called **informational or transactional queries,** where I want to learn about apple pies, or maybe I want to buy a new wireless data plan
- For informational and transactional queries, you might be willing to forgive a web search engine that doesn't give you the best apple pie recipe because there are a lot of good apple pie recipes on the Internet.
- There's a different type of web search query called **navigational queries**, where the user has a very clear intent, a very clear desire to go to Stanford.edu, or Reddit.com, or YouTube.com.
- When a user has a very clear navigational intent, they will tend to be very unforgiving if a web search engine does anything other than return Stanford.edu as the Number one ranked results
- The search engine that doesn't give the right results will quickly lose the trust of its users.
- Navigational queries in this context are a disproportionately important set of examples and if you have a learning algorithm that improves your average test set accuracy for web search but messes up just a small handful of navigational queries, it may not be acceptable for deployment.
- The challenge, of course, is that average test set accuracy tends to weigh all examples equally, whereas in web search, some queries are disproportionately important.
- Now one thing you could do is try to give these examples a higher weight. That could work for some applications, but in my experience, **just changing the weights of different examples doesn't always solve the entire problem**

Performance on key slices of the dataset

Example: ML for loan approval

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

Example: Product recommendations from retailers

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

- Let's say you've built a machine learning algorithm for loan approval to decide who is likely to repay a loan, and thus recommend approving certain loans for approval.
- For such a system, you will probably want to make sure that your system does not unfairly discriminate against loan applicants according to their ethnicity, gender, maybe their location, their language, or other protected attributes.
- Even if a learning algorithm for loan approval achieves high average test set accuracy, it would not be acceptable for production deployment if it exhibits an unacceptable level of bias or discrimination.
- The A.I. community has had a lot of discussion about fairness to individuals, and rightly so because this is an important topic we have to address and do well on
- Let's say you run an online shopping website where you advocate and sell products from many different manufacturers and many different brands of retailers.
- You might want to make sure that your system treats fairly all major user, retailer, and product categories. For example, even if a machine learning system has high average test set accuracy, maybe it recommends better products on average. If it gives really irrelevant recommendations to all users of one ethnicity, that may be unacceptable
- Or if it always pushes products from large retailers and ignores the smaller brands, it could also be harmful to the business because you may then lose all the small retailers. It would also feel unfair to build a recommender system that only ever recommends products from the large brands and ignores the smaller businesses

Rare classes

Skewed data distribution
99% negative   1% positive

print("0")

Accuracy in rare classes

Condition	Performance
Effusion	0.901
Edema	0.924
Mass	0.909
Hernia	0.851

10,000 → Effusion
~100 → Hernia

Input
Chest X-Ray Image

CheXNet
121-layer CNN

Output
Pneumonia Positive (85%)
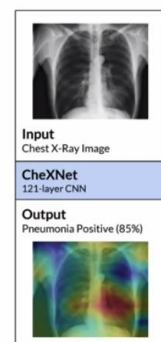
- Next is the issue of rare classes, and specifically that of **skewed data distributions**.
- In medical diagnosis, it's not uncommon for many patients not to have a certain disease,
- If you have a data set which is 99 percent negative examples, it could be because 99 percent of the population doesn't have a certain disease, and only one percent positive.

- You can achieve very good tested accuracy by just writing a program that just says print "0", without having to train a huge neural network that does the same thing
- Closely related to the issue of skewed data distributions is **accuracy** on rare classes.
- I was working on diagnosis of chest X-rays, and using deep learning models to spot different conditions
- Effusion is a common medical condition, where we had about 10,000 images and so, thus we were able to achieve a high level of performance
- However, for the much rarer condition hernia, we had only about a hundred images, therefore performance was much worse.
- It turns out that from a medical standpoint, it is not acceptable for diagnosis system to ignore obvious cases of hernia. Because this was a relatively rare class, the overall average tested accuracy of the algorithm was not that bad
- In fact, the algorithm could have completely ignored all cases of hernia and it would have had only a modest impact on this average test accuracy
- Because cases of hernia were rare, the algorithm could pretty much ignore it without hurting this average tested accuracy that much, especially if average test set accuracy gives equal weight to every single example in the test set.

Unfortunate conversation in many companies

MLE: "I did well on the test set!"

Product Owner: "But this doesn't work for my application"

MLE: "But... I did well on the test set!"

- I have heard pretty much this exact same conversation too many times in too many companies and the conversation goes like this, a machine learning engineer says,
- "I did well in the test set!" this works, and the business owner says, "but this doesn't work for my application" and the machine learning engineer replies, "but I did well on the test set!"
- If you ever find yourself in this conversation, don't get defensive.
- We as a community have built lots of tools for doing well on the test set, and that's to be celebrated. But we often need to go beyond that because just doing well on the test set
- Doing well on the test set isn't enough for many production applications.
- When I'm building a machine learning system, I view it as my job not just to do well on the test set, but to produce a machine learning system that **solves the actual business or application needs**, and I hope you take a similar view as well.

Establish a baseline

- When starting work on a machine learning project, one of the most useful first step to take is to establish a baseline. It is usually only after you've established a baseline level of performance that you can then have tools to efficiently improve on that baseline level.

Establishing a baseline level of performance

🗣️ Speech recognition example:

Type	Accuracy	Human level performance	
Clear Speech	94% ⟶	95%	1 %
Car Noise	89% ⟶	93%	4 %
People Noise	87% ⟶	89%	2 %
⟶ Low Bandwidth	70% ⟶	70%	~ 0 %

- Let's say you've established that there are four major categories of speech in your data.
- With your accuracy on these four categories of speeches as 94, 89, 87, and 70 percent accuracy respectively, you might be tempted to focus our attention on low bandwidth audio (lowest % accuracy)
- But before leaping to that conclusion, it'd be useful to establish a baseline level of performance on all four of these categories. You can do that by asking some human transcriptionists to label your data and measuring their accuracy.
- This determines what is the human level performance (HLP) on these four categories of speech
- In this example, we find that if we can improve our performance on clear speech up to human level performance, looks like there's a potential for a one percent improvement there. If we can raise our performance up to human level performance on audio of car noise in the background, that would be four percent improvement, and zero percent improvement on low bandwidth audio.
- With this analysis, we realized that the low bandwidth audio was so garbled, even humans can't recognize what was said. Thus, it may not be that fruitful to work on that.
- Instead, it may be more fruitful to focus our attention on improving speech recognition with car noise in the background.
- Using HLP gives you a point of comparison or a baseline that helps you decide where to focus your efforts on car noise data rather than on low bandwidth data.

Unstructured and structured data

Unstructured data		Structured data			
Image	[image]	User ID	Purchase	Number	Price
		3421	Blue shirt	5	$20
Audio	[audio icon]	612	Brown shoes	1	$35
Text	This restaurant was great!	Product ID	Product name	Inventory	
		385	Football	158	
		477	Cricket bat	23	

HLP

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

- Unstructured data refers to data sets like images, maybe pictures of cats or audio, like our speech recognition example or natural language
- **Unstructured data** tends to be data that **humans are very good** at interpreting (i.e. understanding images and audio)
- Because humans are so good at unstructured data tasks, measuring human level performance or HLP, is often a good way to establish a baseline if you are working on unstructured data.
- In contrast, structured data are the giant databases you might have, such as if you run an e-commerce website, with the data showing which users purchased at what time and what price.
  Humans are not as good at looking at data like this to make predictions. We certainly didn't evolve to look at giant spreadsheets.
- Human level performance is usually a less useful baseline for structured data applications.
- Machine learning developments best practice is quite different, depending on whether you're working on unstructured data or structured data problem.
- Keeping in mind this difference, let's take a look at some ways to establish baselines for both of these types of problems

Ways to establish a baseline

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

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

- Another way to establish a baseline is to do a **literature search for state-of-the-art,** or look at open source results to see what others report
- For example, if you're building a speech recognition system and others report a certain level of accuracy on data that's similar to yours, then that may give you a starting point.
- Using open-source, you may also consider running just a quick-and-dirty implementation, which could start to give you a sense of what may be possible.
- Finally, if you already have a machine learning system running for your application, then the performance of your previous system, performance of your older system can also help you establish a baseline that you can then aspire to improve on.
- In some cases, such as if you're using human level performance, especially on unstructured data problems, this baseline can also give you a sense of what is the irreducible error or what is Bayes error. In other words, what is the best that anyone could possibly hope for in terms of performance on this problem?
- For example, we may realize that the low bandwidth audio is so bad that it is just not possible to have more than 70% accuracy
- I've seen business teams push a machine learning team to guarantee that the algorithm will be 80% accurate or even 99% accurate, before the team even had a chance to establish a rough baseline
- This, unfortunately, puts the machine learning team in a very difficult position. If you are in that position, I would urge you to consider pushing back and **asking for time to establish a rough baseline level of performance** before giving a more firm prediction about how accurate the machine learning system can eventually get to be.

Getting started on modeling

- Literature search to see what's possible (courses, blogs, open-source projects).

- Find open-source implementations if available.

- A reasonable algorithm with good data will often outperform a great algorithm with no so good data.

- In order to get started on this first step of coming of the model, here are some suggestions.
- When I'm starting on a machine learning project, I almost always start with a quick literature search to see what's possible, so you can look at online courses, look at blogs, and look at open source projects.
- My advice to you if your goal is to build a **practical production** system and not to do research is, don't obsess about finding the latest, greatest algorithm.
- Instead, spend half a day, maybe a small number of days reading blog posts and pick something reasonable that lets you get started quickly. If you can find an open source implementation, that can also help you establish a baseline more efficiently.
- For many practical applications, **a reasonable algorithm with good data will often do just fine and will in fact outperform a great algorithm with not so good data.**
- Don't obsess about taking the algorithm that was just published in some conference last week, which is the most cutting edge algorithm. Instead find something reasonable, find a good open source implementation and use that to get going quickly.
- Because being able to get started on this first step of this loop, can make you more efficient in iterating through more times, and that will help you get to good performance more quickly.

Deployment constraints when picking a model

Should you take into account deployment contraints when picking a model?

Yes, if baseline is already established and goal is to build and deploy.

No (or not necessarily), if purpose is to establish a baseline and determine what is possible and might be worth pursuing.

- If the baseline is already established, and you're relatively confident that this project will work (i.e. goal is to build and deploy a system), then yes, you should take deployment constraints such as compute constraints into account
- **But** if you have not yet even established a baseline, or if you are in a stage of the project where your goal is to just establish a baseline and determine if this project is even worth pursuing for the long term, then it might be okay to ignore deployment constraints, and just find some open source implementation and try it out to see what might be possible.
- This is even if that open source implementation is so computationally intensive that you know you will never be able to deploy that. It is okay to focus on efficiently establishing the baseline first.

Sanity-check for code and algorithm

- Try to overfit a small training dataset before training on a large one.

- Example #1: Speech recognition

  audio   transcript
  x ➡ y   ⎵ ⎵ ⎵ ⎵ ⎵ ⎵

- Example #2: Image segmentation

- Example #3: Image classification
  10,000
  10 , 100

- When trying out a learning algorithm for the first time, before running it on all your data,
- I would urge you to run a few quick sanity checks for your code and your algorithm.
- For example, I will usually try to overfit a very small training dataset before spending hours or sometimes even overnight or days training the algorithm on a large dataset.
- For example, I was once working on a speech recognition system where the goal was to input audio and have a learning algorithm output a transcript. When I trained my algorithm on just one example, one audio clip, it outputs multiple spaces (blanks)
- Clearly it wasn't working and because my speech system couldn't even accurately transcribe one training example, so there wasn't much point to spending hours training it on a giant training set.
- For image segmentation, if your goal is to take as input pictures like this and segment out the cats in the image, before spending hours training your system on hundreds or thousands of images, a worthy sanity check would be to feed it just one image and see if it can at least overfit that one training example, before scaling up to a larger dataset.
- The advantage of this is you may be able to train your algorithm on one or a small handful of examples in just minutes or maybe even seconds and this lets you find bugs much more quickly.
- Finally, for image classification problems, it might be worthwhile to quickly train your algorithm on a small subset of just 10 or maybe 100 images, because you can do that quickly. If your algorithm can't even do well on 100 images, then it's clearly not going to do well on 10,000 images.

Error Analysis Example

- The first time you train a learning algorithm, you can almost guarantee that it won't work not the first time out. So I think of the heart of the machine learning development process is error analysis
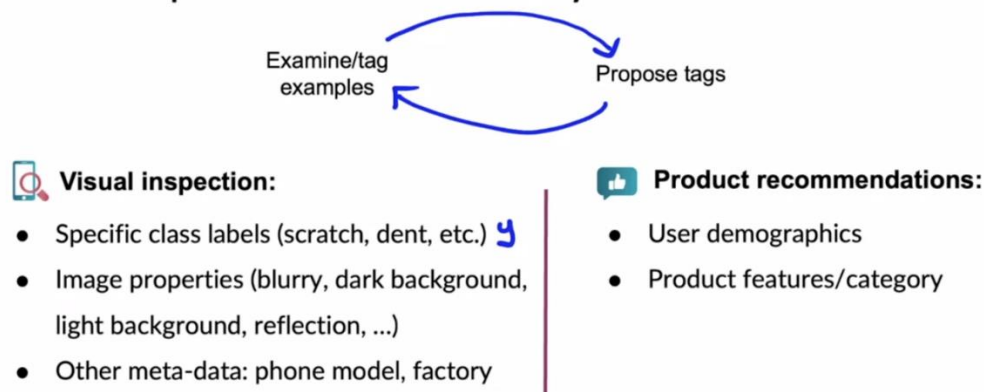
Speech recognition example

Example	Label	Prediction	Car noise	People noise	Low bandwidth
1	"Stir fried lettuce recipe"	"Stir fry lettuce recipe"	1		
2	"Sweetened coffee"	"Swedish coffee"		1	1
3	"Sail away song"	"Sell away some"		1	
4	"Let's catch up"	"Let's ketchup"	1	1	1

- When I'm carrying out error analysis, this is pretty much what I would do myself in a spreadsheet to get a handle on the errors of the speech system.
- You might listen to maybe 100 mislabelled examples from your dev set from the development set.
- When you listen to this example, if this example has car noise in the background, you can then make an annotation in your spreadsheet to indicate that this example had car noise.

- I literally fire up a spreadsheet program like Google sheet or Excel or on a Mac, and do it like this in the spreadsheet.
- This process helps you understand whether the categories, as denoted by tags that may be the source of more of the errors and may be worthy of further effort and attention.
- Until now, error analysis has typically been done via a manual process, say, in the Jupyter notebook or tracking errors in spreadsheet.
- I still sometimes do it that way and if that's how you're doing it too, that's fine. But there are also emerging MLOps tools that making this process easier for developers.
- For example, when my team at LandingAI works on computer vision applications, the whole team uses Landing Lens, which makes this much easier than the spreadsheet.

Iterative process of error analysis

Examine/tag examples → Propose tags

Visual inspection:
- Specific class labels (scratch, dent, etc.)
- Image properties (blurry, dark background, light background, reflection, ...)
- Other meta-data: phone model, factory

Product recommendations:
- User demographics
- Product features/category

- Training a model and deploying a model are iterative processes. So it should come as no surprise that error analysis is also an iterative process
- A typical process is that you might examine and tag some set of examples with an initial set of tags such as car noise and people noise. And based on examining this initial set of examples, you may come back and say you want to propose some new tags.
- With the new tags, you can then go back to examine and tag even more examples.
- Take visual inspection as an example. You know, the problem of finding defects in smart phones. Some of the tags could be specific **class labels**, like scratch or dents. Some of the tags could be **image properties**, such as whether this picture of the phone is blurry? Is it against the dark background or a light background?
- The tags could also come from other forms of **metadata**. Examples are what is the film model? What is the factory which is the manufacturing line that captured the specific image?
- And the goal of this type of process is to try to come up with a few categories where you
- could productively improve the algorithm
- Another example is product recommendations for an online e commerce site. You might look at what products a system is recommending to users and find the clearly incorrect or irrelevant recommendations, and try to figure out if there are specific user demographics
- For example, are we really badly recommending products to younger women or to older men or to something else? Or are there specific product features or specific product categories where the recommendations are particularly poor.
- By alternatively brainstorming and applying such tags, you can come up with a few ideas for categories of data that we're trying to improve your algorithm's performance on.

Useful metrics for each tag

- What fraction of errors has that tag? **12%**
- Of all data with that tag, what fraction is misclassified? **18%**
- What fraction of all the data has that tag?
- How much room for improvement is there on data with that tag?

- As you go through these different tags, here are some useful numbers to look at. First what fraction of errors have that tag? For example, if you listen to 100 audio clips and find that 12% of them were labelled with the car noise type, then that gives you a sense of how important is it to work on car noise.
- It tells you also that even if you fix all of the car noise issues, the performance may improve only by 12%, which is actually not bad.
- Or you can ask all the data with that tag what fraction is misclassified? You might focus your attention on tagging the misclassified examples.
- You can ask of all the data of that tag what fraction is misclassified? So for example, if you find that of all the data with car noise, 18% of it is wrongly transcribed. That tells you that the performance on data with this type of tag has only a certain level of accuracy and tells you how hard these examples with car noise really are.
- You might also ask what fraction of all the data has that tag. This tells you how important relative to your entire data set are examples with that tag.
- Lastly, how much room for improvement is there on data with that tag? One example you've already seen is to measure human level performance on data with that tag.
- By brainstorming different tags, you can segment your data into different categories, and then use questions like these to try to decide what to prioritize working on.

Prioritizing what to work on

Prioritizing what to work on

Type	Accuracy	Human level performance	Gap to HLP	% of data	
Clean Speech	94%	95%	1%	60%	→ 0.6%
Car Noise	89%	93%	4%	4%	→ 0.16%
People Noise	87%	89%	2%	30%	→ 0.6%
Low Bandwidth	70%	70%	0%	6%	→ ~0%

- Rather than deciding to work on car noise because the gap to HLP is bigger, one other useful factor to look at is what's the **percentage of data with that tag**?
- Let's say that 60 percent of your data is clean speech, four percent is data with car noise, 30 percent has people noise, and six percent is low bandwidth audio.
- This tells us that if we could take clean speech and raise our accuracy from 94-95 percent on all the clean speech, then multiplying one percent with 60 percent just tells us that our **overall speech system would be 0.6 percent more accurate**
- On the car noise, if we can improve the performance by four percent on four percent of the data, multiplying that out, that gives us a 0.16 percent improvement

- Whereas previously, we had said there's a lot of room for improvement in car noise, in this slightly richer analysis, we see that because people noise and clean speech accounts for such a large fraction of the data, it may be more worthwhile to work on either of them, because there's actually larger potential for improvements in both of those than for speech with car noise.

Prioritizing what to work on

Decide on most important categories to work on based on:

- How much room for improvement there is.
- How frequently that category appears.
- How easy is to improve accuracy in that category.
- How important it is to improve in that category.

- To summarize, when prioritizing what to work on, you might decide on the most important categories to work on based on, how much room for improvement there is, such as, compared to human-level performance or according to some baseline comparison.
- How frequently does that category appear? You can also take into account how easy it is to improve accuracy in that category.
- Finally, decide how important it is to improve performance on that category. For example, you may decide that improving performance with car noise is especially important, because when you're driving, you have a stronger desire to do search, like search on maps and find addresses without needing to use your hands when your hands are holding the steering wheel.
- There is no mathematical formula that will tell you what to work on. But by looking at these factors, I hope you'd be able to make more fruitful decisions.

Adding/improving data for specific categories

For categories you want to prioritize:

- Collect more data
- Use data augmentation to get more data
- Improve label accuracy/data quality

Type	Accuracy	Human level performance	Gap to HLP	% of data
Clean Speech	94%	95%	1%	60%
Car Noise	84%	93%	4%	4%
People Noise	87%	89%	2%	30%
Low Bandwidth	70%	70%	0%	6%
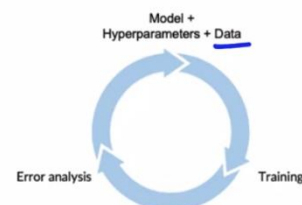
- Once you've decided that there's a category, or maybe a few categories where you want to improve the average performance, one fruitful approach is to consider adding data or improving the quality of that data for that small handful of categories.
- For example, if you want to improve performance on speech with car noise, you might go out and collect more data with car noise. Or if you have a way of using data augmentation to get more data from data category, that will be another way to improve your algorithm's performance.

- In machine learning, we always would like to have more data, but going out to collect more data generically, can be very time-consuming and expensive. By carrying out an analysis like this, you can be **much more focused** in exactly what types of data you need to collect to improve your learning algorithm's performance.
- Because if you decide to collect more data with car noise or maybe people noise, you can be much more specific in going out to collect more of just that data or using data augmentation, without wasting time trying to collect more data from a low bandwidth cell phone connection.

Skewed datasets

Examples of skewed datasets

Manufacturing example

99.7% no defect $y=0$

0.3% defect $y=1$

$print("0")$

99.7%

Medical Diagnosis example: 99% of patients don't have a disease

Speech Recognition example: In wake word detection, 96.7% of the time wake word doesn't occur

- Given a manufacturing company that makes smartphones: If 99.7 percent have no defect (labelled y equals 0) and only a small fraction is labelled y equals 1, then printing 0 all the time will still give 99.7% accuracy, which is not a very impressive learning algorithm.
- For medical diagnosis, if 99 percent of patients don't have a disease, then an algorithm that predicts no one ever has a disease will have 99 percent accuracy
- For speech recognition, if you're building a system for wake word detection, sometimes also called trigger word detection, most of the time that special wake word or trigger word is not being spoken by anyone at that moment in time.
- When I built wake word detection systems, the data sets were actually quite skewed. One of the data sets I used had 96.7% negative examples and 3.3% positive examples.
- When you have a skewed data set like this, accuracy is not that useful a metric to look at, because printing zero can still get you very high accuracy.

Confusion matrix: Precision and Recall

	Actual	
Predicted	$y=0$	$y=1$
$y=0$	905 TN	18 FN
$y=1$	9 FP	68 TP
	↳914	↳86

TN: True Negative
TP: True Positive
FN: False Negative
FP: False Positive

$$Precision = \frac{TP}{TP+FP} = \frac{68}{68+9} = 88.3\%$$

$$Recall = \frac{TP}{TP+FN} = \frac{68}{68+18} = 79.1\%$$

- Instead, it's more useful to build something called the confusion matrix. A confusion matrix is a matrix where one axis is labelled with the actual label, is the ground truth label, y equals 0 or y equals 1, and whose other axis is labelled with the prediction.
- Here we have a pretty skewed data set where out of 1000 examples, there were 940 negative examples and just 86 positive examples i.e. 8.6 percent positive, 91.4 percent negative.
- The metrics of precision and recall are more useful than raw accuracy when it comes to evaluating the performance of learning algorithms on skewed data sets.

What happens with print("0")?

$$Precision = \frac{TP}{TP + FP} = \frac{0}{0 + 6}$$

$$Recall = \frac{TP}{TP + FN} = \frac{0}{0 + 86} = 0\%$$
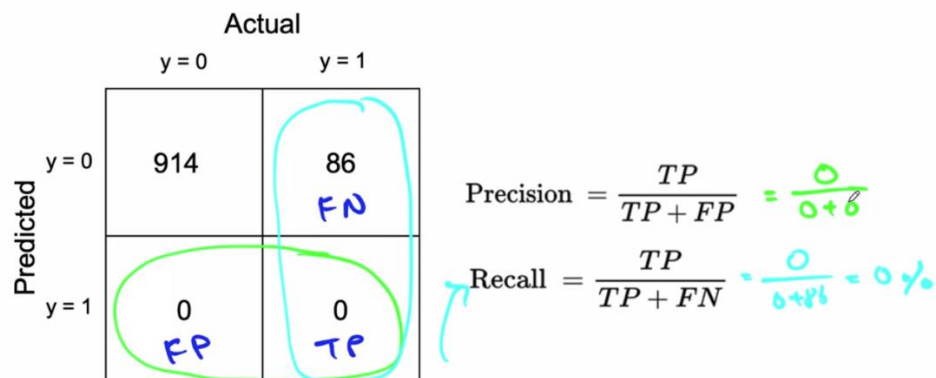
- Taking this example of where we had 914 negative examples and 86 positive examples, if the algorithm outputs zero all the time, this is what the confusion matrix will look like
- The 0 algorithm achieves zero percent recall, which gives you an easy way to flag that this is not detecting any useful, positive examples.
- The learning algorithm with even with a high value of precision is not that useful usually if this recall is so low.

Combining precision and recall – F_1 score

	Precision (P)	Recall (R)	F_1
Model 1	88.3	79.1	83.4% ←
Model 2	97.0	7.3	13.6%

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

- Sometimes you have one model with a better recall and a different model with a better precision. How do you compare two different models?
- There's a common way of combining precision and recall using this formula, which is called the F1 score. One intuition behind the F1 score is that you want an algorithm to do well on both precision and recall, and if it does worse on either precision or recall, that's pretty bad.

pg. 15

- In mathematics, this is technically called a harmonic mean between precision and recall, which is like taking the average but placing more emphasis on whichever is the lower number.
- For your application, you may have a different weighting between precision and recall, and so F1 isn't the only way to combine precision and recall, it's just one metric that's commonly used for many applications.

## Multi-class metrics

Classes: Scratch, Dent, Pit mark, Discoloration

Defect Type	Precision	Recall	F_1
Scratch	82.1%	99.2%	89.8%
Dent	92.1%	99.5%	95.7%
Pit mark	85.3%	98.7%	91.5%
Discoloration	72.1%	97%	82.7%

- So far, we've talked about binary classification problem with skewed data sets.
- It turns out to also frequently be useful for multi-class classification problems.
- If you are detecting defects in smartphones, you may want to detect scratches on them, dents, pit marks, or discolouration of cell phone's LCD screen
- Maybe all four of these defects are actually quite rare that you might want to develop an algorithm that can detect all four of them.
- One way to evaluate how your algorithm is doing on all four of these defects, each of which can be quite rare, would be to look at precision and recall of each of these four types of defects individually.
- You find in manufacturing that many factories will want high recall because you really don't want to let a defective phone go out
- But if an algorithm has slightly lower precision, that's okay, because through a human re-examining the phone, they will hopefully figure out that the phone is actually okay, so **many factories will emphasize high recall**.
- By combining precision and recall using F1 as follows, this gives you a single number evaluation metric for how well your algorithm is doing on the four different types of defects, and can also help you benchmark to human-level performance and also prioritize what to work on next.
- Instead of accuracy on scratches, dents, pit marks, and discolorations, using F1 score can help you to prioritize the most fruitful type of defect to try to work on.
- The reason we use F1 is because maybe all four defects are very rare, thus accuracy would be very high even if the algorithm was missing a lot of these defects.
- These tools will help you both evaluate your algorithm as well as prioritize what to work on, both in problems with skewed data sets and for problems with multiple rare classes.

Performance auditing

- Even when you're learning algorithm is doing well on accuracy or F1 score, or some appropriate metric, it is often worth one last performance audit before you push it to production.
- And this can sometimes save you from significant post deployment problems.

Performance auditing

Model +
Hyperparameters + Data

Error analysis

Training
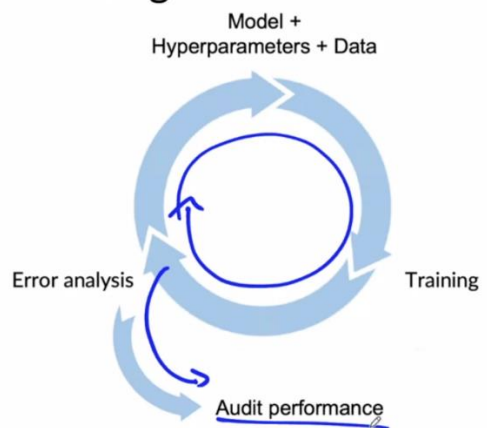
Audit performance

- After you've gone around this cycle multiple times to develop a good learning algorithm. It's worthwhile auditing this performance one last time

Auditing framework

Check for accuracy, fairness/bias, and other problems.

1. Brainstorm the ways the system might go wrong.
   - Performance on subsets of data (e.g., ethnicity, gender).
   - How common are certain errors (e.g., FP, FN).
   - Performance on rare classes.

2. Establish metrics to assess performance against these issues on appropriate slices of data.

3. Get business/product owner buy-in.

- Here's a framework for how you can double check your system for accuracy, fairness, bias and other possible problems.
- Step one is to brainstorm the different ways the system might go wrong.
- For example, does the algorithm perform sufficiently well on different subsets of the data, such as individuals of a certain ethnicity or individuals of different genders? Does the algorithm make certain errors such as false positives and false negatives? How does it perform on rare and important cases?
- You might then establish metrics to assess the performance of your algorithm against these issues. One very common design patterns you see is that you often be evaluating performance on slices of the data.
- So rather than evaluating performance on your entire dev set, you may be taking out all of the individuals of a certain ethnicity, all the individuals of a certain gender or all of the examples where there is a scratch defect on the smartphone (also called slices of the data)
- After establishing appropriate metrics, MLOps tools can help trigger an automatic evaluation for each model to audit this performance. For instance, Tensorflow has a package called Tensorflow model analysis (TFMA) that computes detailed metrics on new models on different slices of data.
- And as part of this process, I would also advise you to get buy-in from the business or product owner, letting them know that these are the most appropriate set of problems to worry about, and that there is a reasonable set of metrics to assess against these possible problems.

Speech recognition example

1. Brainstorm the ways the system might go wrong.
   - Accuracy on different genders and ethnicities.
   - Accuracy on different devices.
   - Prevalence of rude mis-transcriptions.

   GAN   gun   gang

2. Establish metrics to assess performance against these issues on appropriate slices of data.
   - Mean accuracy for different genders and major accents.
   - Mean accuracy on different devices.
   - Check for prevalence of offensive words in the output.

- If you build a speech recognition system, you might then brainstorm the way the system might go wrong. So one thing I've looked at was the accuracy on different genders and different ethnicities.
- One type of analysis I've also done before is to carry out analysis of our accuracy depending on the perceived accent of the speaker because we want to understand if the speech systems performance was a huge function of the accent of the speaker
- You might also worry about the accuracy on different devices because different devices may have different microphones.
- Another issue is the prevalence of rude mistranscriptions. Here's one example of something that actually happened to some of DeepLearning.AI's courses. One of our instructors, Laurence Maroney was talking about GANs, generative adversarial networks, but the transcription system was mistranscribing GANs because this unfortunately is not a common word in English language. And so, the subtitles had a lot of references to gun and gang, which were mistranscriptions of what the instructor actually said, which is GANs.
- And so we need to pay special attention to avoiding mistranscriptions that resulted in the speech system thinking someone said a swear word, when maybe they didn't actually say that swear word
- Next, you can then establish metrics to assess performance against these issues on the appropriate slices of data. For example, you can measure the mean accuracy of the speech system for different genders and for different accents represented in the data set and also check for accuracy on different devices and check for offensive or roots words in the output.
- I find that the ways a system might go wrong turns out to be very problem dependent.
- Different industries, different tasks will have very different standards. Those standards are still continuing to evolve in AI and in many specific industries.
- So I would advise you to do a search for your industry to see what is acceptable and to keep current with standards of fairness and all of our growing awareness for how to make our systems fairer and less biased.
- It would be good to have a team or sometimes even external advisors to help you brainstorm, so as to reduce the risk of you or your team being caught by something that you hadn't thought of.

Data-centric AI development

| Model-centric view	Data-centric view
Take the data you have, and develop a model that does as well as possible on it.	The quality of the data is paramount. Use tools to improve the data quality; this will allow multiple models to do well.
Hold the data fixed and iteratively improve the code/model.	*Hold the code fixed and iteratively improve the data.*

- With a model centric view of AI developments, you would take the data you have and then try to work really hard to develop a model that does as well as possible on the data
- This is because a lot of academic research on AI was driven by researchers, downloading a benchmark data set, and trying to do well on that benchmark.
- Most academic research on AI is model centric, because the benchmark data set is a fixed quantity. In this view, model centric development, you would hold the data fixed and iteratively improve the code or the model.
- There's still an important role to play in trying to come up with better models, but that's a different view of AI developments which I think is more useful for many applications.
- It is to shift a bit from a model centric to more of a data centric view.
- In this view, we think of the quality of the data as paramount, and you can use tools such as error analysis or data augmentation to systematically improve the data quality.
- For many applications, I find that if your data is good enough, there are multiple models that will do just fine.
- In this view, you can instead hold the code fixed and iteratively improve the data. There's a role for model centric development, and there's a role for data centric development.
- If you've been used to model centric thinking for most of your experience with machine learning, I would urge you to consider taking a data centric view as well

A useful picture of data augmentation
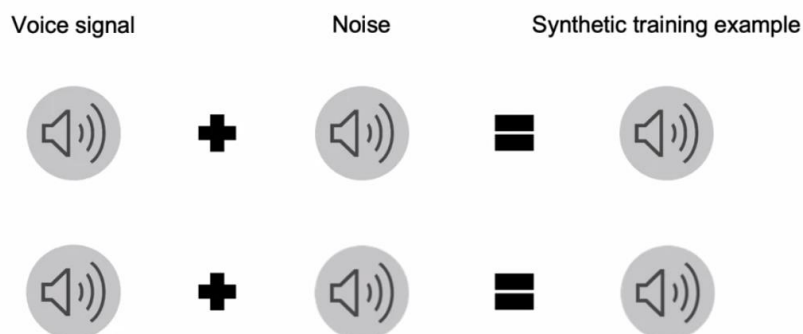

Speech recognition example
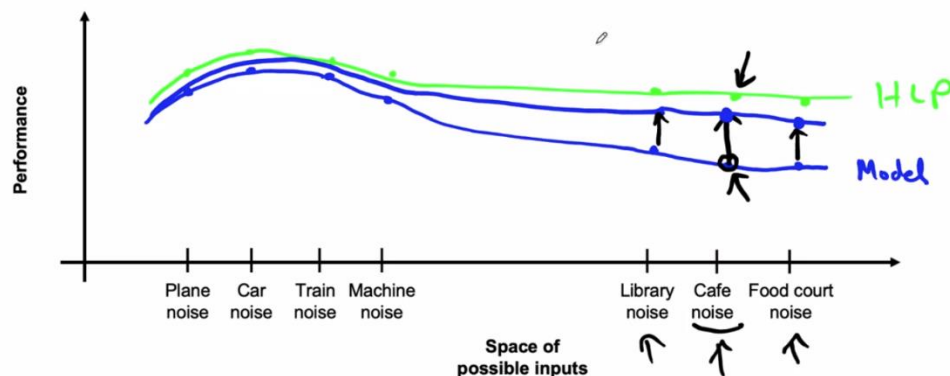
- In this diagram, the vertical axis represents performance, say accuracy. And on the horizontal axis is a conceptual kind of a thought experiment type of axis, which represents the space of possible inputs.
- For example we have speech with car noise and plane noise and train noise, all of which are quite similar with machine noise (e.g. washing machine, air conditioners).
- On the other side you have speech with cafe noise, library noise or food court, and these are more similar to each other than the other types of mechanical noise.
- Your system will have different levels of performance on these different types of input.
- Let's say the performance on data with library noise, cafe noise, food court noise does worse than that of the mechanical noises
- And so I think of all of these as being a curve, like a one dimensional piece of rubber band/sheet that shows how accurate your speech system is as a function of the type of input it gets. This is represented by a blue curve.
- A human will have some other level of performance on these different types of data (represented by green line), meaning that human level performance is represented via some other curve.
- So this gap between the two curves represents an opportunity for improvement.
- As you perform data augmentation on specific types of data, imagine you are grabbing a hold of a point on the blue rubber band and pulling it upward like so.
- What that will tend to do is pull up this rubber sheet in the **adjacent region** as well. So if performance on cafe noise goes up, probably performance on the nearby points will go up too
- It turns out that for unstructured data problems, pulling up one piece of this rubber sheet is unlikely to cause a different piece of the rubber sheet to dip down really far below.
- Instead, pulling up one point causes nearby points to be pulled up quite a lot and far away points may be pulled up a little bit, or if you're lucky, maybe more than a little bit.
- But when I'm planning how to improve my learning algorithm's performance and where I hope to get it to, and getting more data in those places to iteratively pull up with those pieces or those parts of the rubber sheet to get them closer to human level performance.
- Error analysis will tell you the location of this new biggest gap, which may then be worth your effort, to collect more data on and therefore to try to pull up one piece at a time.

- Data augmentation can be a very efficient way to get more data, especially for unstructured data problems such as images, audio, maybe text.
- But when carrying out data augmentation, there're a lot of choices you have to make. What are the parameters? How do you design the data augmentation setup?

Data augmentation example

Voice signal		Noise		Synthetic training example
🔊	✚	🔊	＝	🔊
🔊	✚	🔊	＝	🔊

- Take speech recognition. Given an audio clip, and you take background cafe noise, and add these two audio clips together (i.e. literally, take the two waveforms and sum them up), you can create a synthetic example that sounds like someone is speaking in a noisy cafe
- When carrying out data augmentation, there're a few decisions you need to make, such as what is the type of background noise and how loud the background noise should be relative to the speech.
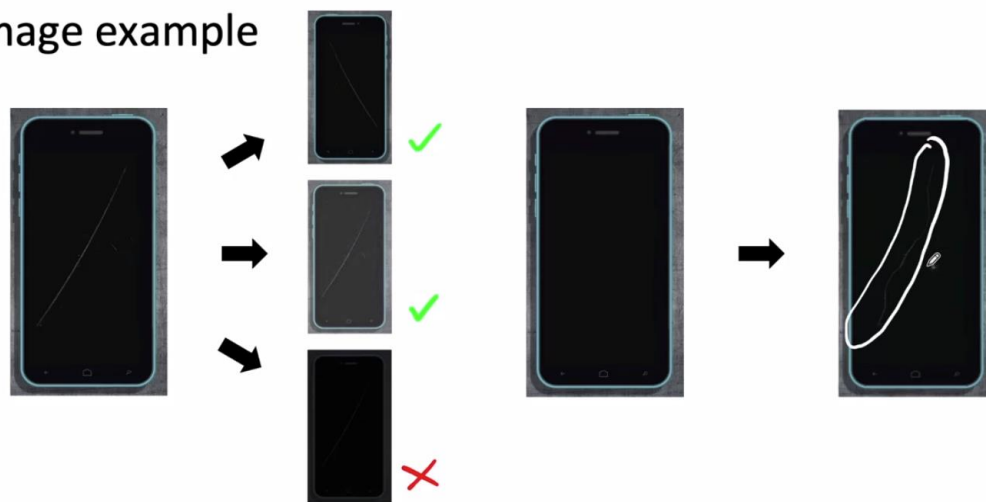
Data augmentation

Goal:

Create realistic examples that (i) the algorithm does poorly on, but (ii) humans (or other baseline) do well on

Checklist:

- Does it sound realistic?
- Is the x → y mapping clear? (e.g., can humans recognize speech?)
- Is the algorithm currently doing poorly on it?

- The goal of data augmentation is to create examples that the learning algorithm can learn from.
- As a framework for doing that, I encourage you to think of how you can create realistic examples that the algorithm **does poorly on**, because if the algorithm already does well in those examples, then there's less for it to learn from.
- But you want the examples to still be ones that a human or maybe some other baseline can do well on, because otherwise, one way to generate examples would be to just create examples that are so noisy that no one can hear what anyone said (which is not helpful)
- You want examples that are hard enough to challenge the algorithm, but not so hard that they're impossible for any human or any algorithm to ever do well on.
- Now, one way that some people do data augmentation is to generate an augmented data set, and then train the learning algorithm and see if the algorithm does better on the dev set.

- They then fiddle around with the parameters for data augmentation and change the learning algorithm again and so on. This turns out to be quite inefficient because every time you change your data augmentation parameters, you need to train your algorithm all over again.
- Instead, using these principles allows you to sanity check that your new data generated using data augmentation is useful without actually having to spend maybe hours or days of training a learning algorithm on that data to verify that it will result in the performance improvement.
- Specifically, here's a checklist you might go through when you are generating new data.
- **One**, does it sound realistic. You want your audio to actually sound like realistic audio of the sort that you want your algorithm to perform on.
- **Two**, is the X to Y mapping clear? In other words, can humans still recognize what was said?
- **Three**, is the algorithm currently doing poorly on this new data. That helps you verify point one.
- If you can generate data that meets these criteria, then that would give you a higher chance that when you put this data into your training set and retrain the algorithm, you will get better results


Image example

- Let's look at one more example, using images this time. Let's say that you have a very small set of images of smartphones with scratches. Here's how you may be able to use data augmentation.
- You can take the image and flip it horizontally. This results in a pretty realistic image, and this could be a useful example to add to your training set.
- Or you could implement contrast changes to actually brighten up the image here, so the scratch is a little bit more visible.
- Or you could try darkening the image, but in this example at the bottom, the image is now so dark that even I as a person can't really tell if there's a scratch there or not.
- Whereas these two examples on top would pass the checklist we had earlier, this example is too dark and would fail that checklists.
- I would try to choose the data augmentation scheme that generates more examples that look like the ones on top and few of the ones that look like the ones here at the bottom.
- In fact, going off the principle that we want images that look realistic, that humans can do well on and hopefully the algorithm does poorly on, you can also use more sophisticated techniques such as take a picture of a phone with no scratches and use Photoshop in order to artificially draw a scratch.
- This technique, literally using Photoshop, can also be an effective way to generate more realistic examples
- I've also used more advanced techniques like GANs to synthesize scratches like these automatically, though I found that techniques to be overkill
- There're simpler techniques that are much faster to implement that work just fine without the complexity of building a GAN to synthesize scratches.

Data iteration loop

Model iteration

Add/improve Data
(holding model fixed)

Error analysis
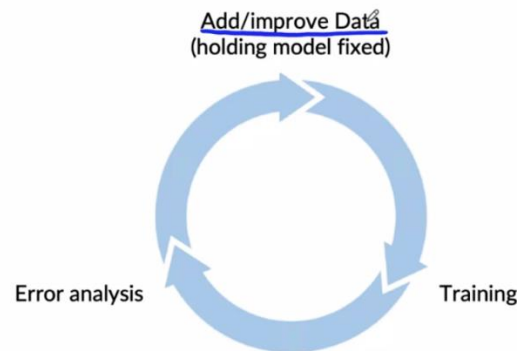
Training

- You may have heard of the term model iteration, which refers to iteratively training a model using error analysis and then trying to decide how to improve the model.
- Taking a data-centric approach AI developments, it's useful to instead use a data iteration loop where you repeatedly take the data and the model, train your learning algorithm, do error analysis, and as you go through this loop, focus on how to add data or improve the data quality
- For many practical applications, taking this data iteration loop approach (along with a robust hyperparameter search) results in faster improvements to your learning algorithm performance, depending on your problem.

Can adding data hurt?

- For a lot of machine learning problems, training sets and dev and test set distribution start off as being reasonably similar.
- But, if you're using data augmentation, you're adding to specific parts of the training set such as adding lots of data with cafe noise (i.e. get data in areas where algorithm is not doing well on)
- So now you're training set may come from a very different distribution than the dev set and the test set. Is this going to hurt your learning album's performance?
- Usually the answer is **no** with some caveats when you're working on unstructured data problems

Can adding data hurt performance?

For unstructured data problems, if:

- The model is large (low bias).
- The mapping x → y is clear (e.g., given only the input x, humans can make accurate predictions).

Then, **adding data rarely hurts accuracy.**

$P(x)$

20% Cate
↳ 50%

- If you are working on an unstructured data problem and if your model is large, such as a neural network that is quite large and has a large capacity and has low bias, then it is not a big issue

- And also if the mapping from x to y is clear (meaning that given only the input x, humans can make accurate predictions y), it turns out that adding accurately labelled data rarely hurts accuracy.
- This is an important observation because adding data through data augmentation or collecting more of one type of data, can change your input data distribution
- Let's say at the start of your problem, 20% of your data had cafe noise. But using augmentation, you added a lot of cafe noise to make up 50%
- So long your model is sufficiently large, it won't stop it from doing a good job on the cafe noise data **as well as** doing a good job on non-cafe noise data.
- In contrast, if your model was **small**, then changing your input data distribution this way may cause it to spend too much of its resources modelling cafe noise settings, and this could hurt this performance on non-cafe noise data.

Photo OCR counterexample

Adding a lot of new "I"s may skew the dataset and hurt performance

- The second problem that could arise is if the mapping from x to y is not clear, meaning given x, the true label of y is very ambiguous. This doesn't really happen much in speech recognition, but let me illustrate this with an example from computer vision.
- This is very rare, so it's not something I would worry about the most practical problems, but let's see why this is important.
- One of the systems I had worked on many years ago was on Google street view images, with the aim of reading house numbers in order to more accurately clear locate houses in Google maps.
- So one of the things that system did was take input pictures like this and figure out the digits.
- So clearly this is a number one (left image) and this is an alphabet I (middle image).
- You don't see a lot of I's in street view images as house numbers rarely have an alphabet I in it.
- We have an algorithm that has very high accuracy on recognizing the number ones, but low accuracy on recognizing I's.
- One thing you might do is add a lot more examples of I's in your training set.
- The problem comes (although rare) is that some images are truly ambiguous (right image). Is this a one or is this an I?
- And if you were to add a lot of new I's to your training set, especially ambiguous examples like these, then that may skew the data sets to have a lot more I's
- This will hurt performance because we know that there are a lot more ones than I's on house numbers.
- It would thus be safer to guess that this is a one rather than that this is an I.
- But if data augmentation skews the data set in the direction of having a lot more I's rather than a lot of ones, which may cause the neural network to guess poorly on ambiguous examples like this.
- So this is one rare example where adding more data could hurt performance. In particular given only an image like this on the right, even a human can't really tell what this is.
- Just to be clear, the example that we just went through together is a pretty rare case and it's quite unusual for data augmentation or adding more data to hurt the performance of your learning algorithm, as long as your model is big enough.

Adding features

- So far, our discussion has focused on unstructured data problems. How about structured data problems? It turns out there's a different set of techniques that's useful for structured data.
- For many structured data problems, creating brand new training examples is difficult, but there's something else you could do, which is to take existing training examples and figure out if there are additional useful features you can add to it.

Structured data

Restaurant recommendation example

Person, Restaurant → NN → Recommend?

Vegetarians are frequently recommended restaurants with only meat options.

Possible features to add?

- Is person vegetarian (based on past orders)?
- Does restaurant have vegetarian options (based on menu)?

- Let me use an example of restaurant recommendations where you want to recommend restaurants to users
- One way to do this would be to have a set of features for each user, and a set of features for each restaurant. These are then fed into some learning algorithms (e.g. a neural network), which then predicts whether or not this is a good recommendation,
- In this particular example, which is a real example, error analysis showed that the system was unfortunately frequently recommending to vegetarians restaurants that only had meat options.
- So this wasn't a good experience for anyone and there was a strong desire to change this.
- Now, I didn't know how to synthesize new examples of users or new examples of restaurants because this application had a fixed pool of users and there are only so many restaurants.
- So rather than trying to use data augmentation to create brand new people or restaurants to feed to the training set, I thought it was **more fruitful to see if there were features to add** to either the person inputs or to the restaurant inputs.
- Specifically one feature you can consider adding is a feature that indicates whether this person appears to be vegetarian. And this doesn't need to be a binary value feature 0/1. It could be features such as a percentage of food ordered that was vegetarian, or some other measures of how likely they seem to be vegetarian.
- And a feature to add on the restaurant side would be whether it has vegetarian options
- For structured data problems, usually you have a fixed set of users or a fixed set of restaurants or fixed set of products, making it hard to use data augmentation or collect new data from new users that you don't have yet on restaurants that may or may not exist.
- Instead, adding features, can be a more fruitful way to improve the performance of the algorithm to fix problems like this one, identify through error analysis.
- Additional features like these, can be hand coded or they could in turn be generated by some learning algorithm, such as having a learning algorithm try to read the menu and classify meals as vegetarian or not

Other food delivery examples

- Only tea/coffee
- Only pizza

What are the added features that can help make a decision?

Product recommendation:

Collaborative filtering ⟶ Content based filtering   cold-start
                                   ↳ features.

- For food delivery examples, we found that there were some users that would only ever order a tea and coffee and some users are only ever order pizza.
- So if the product team wants to improve the experience of these users, a machine learning team might ask what are the additional features we can add (i.e. enrich features) to detect who are the people that only order tea or coffee or who are the people that only ever order the pizza.
- Over the last several years, there's been a trend in product recommendations of a **shift from collaborative filtering approaches to content based filtering approaches.**
- Collaborative filtering approaches is loosely an approach that looks at the user, tries to figure out who is similar to that user and then recommends things to you that people like you also liked.
- In contrast, a content based filtering approach will tend to look at you as a person and look at the description of the restaurant or look at the menu (or other information) of the restaurants, and then see if that restaurant is a good match for you or not.
- The advantage of content based filtering is that even if there's a new restaurant or a new product that hardly anyone else has liked, by actually looking at the description of the restaurant rather than just looking at who else likes the restaurants, you can still quickly make good recommendations.
- This is sometimes also called the **Cold Start Problem** i.e. how do you recommend a brand new product that almost no one else has purchased or like or dislike so far?
- And one of the ways to do that is to make sure that you capture good features for the things that you might want to recommend.
- Unlike collaborative filtering, which requires a bunch of people to look at the product and decide if they like it or not before it can decide whether a new user should be recommended the same product.

Data iteration
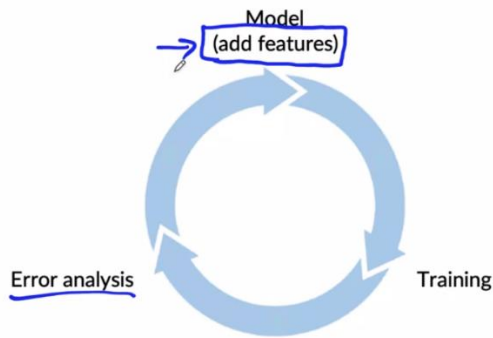
- Error analysis can be harder if there is no good baseline (such as HLP) to compare to.

- Error analysis, user feedback and benchmarking to competitors can all provide inspiration for features to add.

- Data iteration for structured data problems may look like this.
- You start out with some model, train the model and then carry out error analysis.
- Error analysis can be harder on structured data problems if there is **no good baseline** such as human level performance to compare to, and human level performance is hard for structured data because it's really difficult for people to recommend good restaurants even to each other.
- But I found that error analysis, user feedback and benchmarking to competitors can discover ideas for improvement
- Through these methods, if you can identify a certain type of tag associated your data that you want to drive improvement, then you can go back to select some features to add. Examples are like features to figure out who's vegetarian and what restaurants have good vegetarian options
- And because the application may only have a finite list of users and restaurants, the users and restaurants you have maybe all the data you have, adding features to the examples is a more fruitful approach than trying to come up with new users or new restaurants.
- I know that many years ago before the rise of deep Learning, part of the hope for deep learning was that you don't have to hand design features anymore. I think that has for the most part come true for unstructured data problems.
- The larger data set, the more likely it is that a pure end-to-end deep learning algorithm can work.
- But even with the rise of deep learning, if your dataset size isn't massive, designing features, especially for structured data, can still be a very important driver of performance improvements.

Experiment tracking
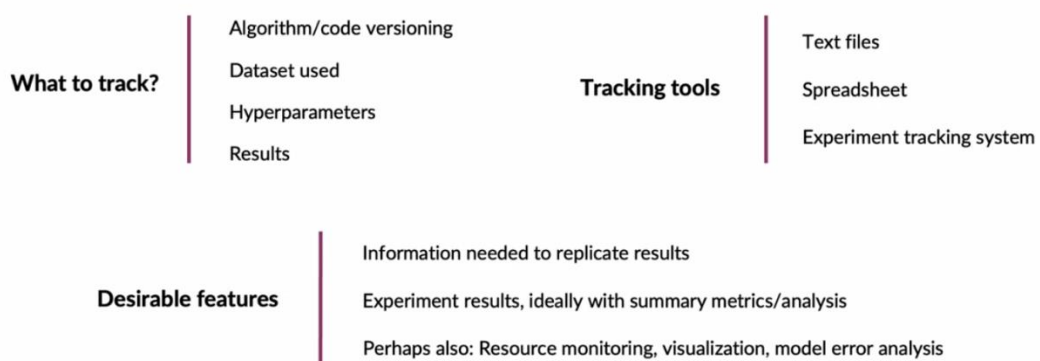
- As you're working to iteratively improve your algorithm, one thing that'll help you be a bit more efficient is to make sure that you have robust experiment tracking.
- When you're running dozens or hundreds or maybe even more experiments, it's easy to forget what experiments you have already run.
- Having a system for tracking your experiments can help you be more efficient in making the decisions on the data or the model or hyperparameters to systematically improve your algorithm's performance.
- When you are tracking the experiments you've run, meaning the models you've trained, here are some things I would urge you to track.
- One, is to keep track of what algorithm you're using and what version of code. Keeping a record of this will make it much easier for you to go back and replicate the experiment
- Second, keep track of the data set you use.
- Third, hyperparameters and fourth, save the results somewhere. This should include at least the high level metrics such as accuracy or F1 score or the relevant metrics, but if possible, it'd be useful to just save a copy of the trained model.
- How can you track these things? Here are some tracking tools you might consider.
- A lot of individuals and sometimes even teams will start off with **text files**.
- When I'm running experiment by myself, I might use a text file to just make a note with a few lines of text per experiment to note down what I was doing.
- This does not scale well, but it may be okay for small experiments.
- A lot of teams then migrate from text files to spreadsheets, especially shared spreadsheets
- Spreadsheets actually scale quite a bit further, especially shared spreadsheets where multiple members of a team may be able to look at.
- Some teams will also consider migrating to a more formal experiment tracking system. The space of experiment tracking systems is still evolving rapidly, and so does a growing set of tools out there. But some examples include Weight&Biases, Comet, MLflow, and SageMaker Studio.
- LandingAI also has its own experiment tracking tool focusing on computer vision and manufacturing applications.
- When I'm trying to use a tracking tool, whether a text file or a spreadsheet or some larger system, here are some of things I look at.
- First is, does it give me all the information needed to replicate the results?
- In terms of replicability, one thing to watch out is if the learning algorithm pulls data off the Internet. Because data off the Internet can change, it can decrease replicability unless you're careful in how your system is implemented.
- Second, tools that help you quickly understand the experimental results of a specific training run, ideally with useful summary metrics and maybe even a bit of an in-depth analysis,
- Finally, some other features to consider, resource monitoring, how much CPU/GPU memory resources do it use? Or tools to help you visualize the trained model or even tools to help you with more in-depth error analysis?
- Rather than worrying too much about exactly which experiment tracking framework to use though, the number one thing I hope you take away from this video is, you got to have some system, even if it's just a text file or just a spreadsheet, for keeping track of your experiments and include as much information as is convenient to include.

From big data to good data

From Big Data to Good Data

Try to ensure consistently high-quality data in all phases of the ML project lifecycle.

Good data:
- Covers important cases (good coverage of inputs x) ←
- Is defined consistently (definition of labels y is unambiguous)
- Has timely feedback from production data (distribution covers data drift and concept drift)
- Is sized appropriately

- A lot of modern AI had grown up in large consumer internet companies with maybe a billion users, and these companies have a lot of data on their users.
- If you have big data like that, by all means, the big data could help the performance of your room tremendously.
- But for many other industries, there's just isn't a billion data points.
- And I think it may be even more important for those applications to focus not just on big data but on good data.
- I found that if you are able to ensure consistently high quality data in all phases in the machine learning project life cycle, it is key to making sure that you have a high performance and reliable machine learning deployment.
- Good data covers the important cases, so you should have good coverage of different inputs x. And if you find out that you don't have enough data, data augmentation can help you get more data get more diverse inputs x and to give you that coverage.
- Good data is also defined consistently with definition of labels y that's unambiguous.
- Good data also has timely feedback from production data. We actually talked about this last week when we were covering the deployment section in terms of having monitoring systems to track concept drift and data drift.
- And finally, you do need a reasonably sized data set.

Readings

- [Establishing a baseline](#)
- [Error analysis](#)
- [Experiment tracking](#)
- Brundage, M., Avin, S., Wang, J., Belfield, H., Krueger, G., Hadfield, G., … Anderljung, M. (n.d.). Toward trustworthy AI development: Mechanisms for supporting verifiable claims*. Retrieved May 7, 2021 http://arxiv.org/abs/2004.07213v2
- Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., & Sutskever, I. (2019). Deep double descent: Where bigger models and more data hurt. Retrieved from http://arxiv.org/abs/1912.02292