Building Machine Learning Systems That Don’t Suck

Program Notes

Session 1

How To Start (Almost) Any Project

Problem framing

1. Most books, courses, and tutorials focus on algorithms, but engineers spend most of their time working with data. Building models is only a small portion of creating working systems.

2. Pablo Picasso once said "Computers are useless. They can only give you answers." Asking the right questions will prevent you from solving the wrong problem or trying to solve the right problem using the wrong solution.

3. Before starting a project, determine what problem you are solving. This question is essential to understanding the project's scope. Focus on a high-level breakdown of the critical path.

4. Before starting a project, determine what problems you are ignoring. The problems you avoid are what differentiate a mediocre solution from an exceptional one.

5. Before starting a project, determine who your customers are and why they care. Avoid the telephone game. Cut the middleman. Identify who cares and why this is important to them. Understanding the problem's importance provides motivation and context for the entire project. It helps prioritize resources and justifies the investment in developing a machine-learning solution. Without clarity on the problem's significance, it's challenging to rally support and commitment.

6. Before starting a project, determine what existing solutions look like. Those who cannot remember the past are condemned to repeat it. Assessing existing solutions allows for learning from past experiences and avoiding reinventing the wheel. It helps identify best practices, pitfalls, and potential opportunities for improvement.

7. Before starting a project, determine how you can measure success. You can't solve a problem without a benchmark to assess progress and guide future improvements.

8. You don't win with better algorithms or better data. You win by framing problems in a way that makes failing impossible.
9. Lawyers don't take cases they can't win. Be like a lawyer. Don't start working on a project until you can build a successful solution.
10. Tesla is a fantastic example of winning by framing their problem in a smart way. Instead of focusing on replacing drivers, they recognized the best way to start was to build a system that augments them.
11. The haystack principle is a powerful mental model when deciding how to solve a problem. Instead of trying to find the needle in a haystack, make the haystack as small as possible.
12. Inversion is a mental model in which you turn a problem upside down to think about it differently. It's a great strategy for identifying new ways to tackle a problem.

How to start a project

1. The first rule of machine learning: Never start a project with machine learning. Instead, start with the simplest solution that could possibly work.
2. Make it work. Make it right. Make it fast.
3. Simple ideas are better than clever ideas. Machine learning is clever and powerful, but is not free. A better alternative is to start using simple heuristics. For example, use a regular expression instead of a Large Language Model, or build a simple function with conditional logic instead of a neural network.
4. In most situations, if you don’t know how to build a good application using a manual approach, machine learning will be a waste of time.
5. Start by doing things that don't scale. Start by identifying opportunities to become the software instead of having to build software.
6. Simplicity is the key to building a faster feedback loop. Early feedback is crucial for validating assumptions, reducing risk, adapting to changes, and avoiding mistakes.

Data collection

1. There are no datasets in the real world. The best way to collect real production data from the users is through a working solution.
2. There are two ways to gather data. You can go and collect it, or you can let the data come to you by building a product that your users use. The latter is the best approach to getting access to real data.
3. Never trust existing datasets to represent real production data. A dataset is a cheap representation of production data. The data you collect and curate will never have the same value as fresh data.
4. The three key elements of a good dataset are quality, diversity, and quantity.
5. More data is not always better—and it’s often worse. You need to maintain data, deal with privacy implications, and pay for processing and storage costs. Better data is better than more data.
6. Understanding the value of additional data is crucial. To determine whether your model has the potential to improve with additional data, you can use a learning curve that shows the relationship between the amount of training data and the model's performance.
7. Selection bias is inevitable and requires a data-centric approach to work around it. Selection bias makes training and production data different.
8. A few common causes of selection bias are time, the location where we collect the data, demographics bias, response bias, and availability bias.
9. An example of selection bias related to time occurs when training data collected in the past is used to make predictions about future events, where the future data may differ significantly.
10. An example of selection bias related to location occurs when training data is collected from one location, but the model is deployed in a different place. This may lead to poor predictions due to different regional characteristics.
11. An example of selection bias related to demographics occurs when we train a model using data from a minority group, and try to make predictions using that model on a general population.
12. An example of selection bias related to response bias occurs when survey response rates are low or when the wording of questions and available answer choices introduce bias, affecting the reliability of the data.

13. An example of selection bias related to availability occurs when the data used to train a model is chosen based on convenience rather than representativeness, which can lead to a model that does not perform well across all scenarios.
14. An example of selection bias related to long tail bias occurs when the mechanisms for collecting data prevent rare events from appearing in the dataset, leading to a model that may not accurately predict these events when they do occur.
15. Collect as much metadata that helps explain your data as possible. This information will become critical to evaluate your model and counter the effects of selection bias.

Labeling strategies

1. Models need ground truth labels to learn.
2. Human annotations consist of subject matter experts providing labels for each sample in a dataset. This produces high-quality labels, but creating manual labels is expensive, and doesn't scale.
3. Many people feel like labeling is beneath them and see it as grunt work, but we can't build a good model without a robust strategy to label data.
4. Some problems have built-in, natural labels. In these problems, we can automatically evaluate the model's predictions without needing manual labels.
5. Some examples of problems with natural labels are predicting the arrival of planes, the price of a stock, or whether it will snow in the future.
6. The lack of labels is one of the biggest bottlenecks when building machine learning systems.
7. Having the wrong labels is worse than having no labels at all. You can't out-train bad labels.
8. Weak supervision can help us scale labeling. We can use it to generate labels automatically by using high-level and often noisier sources of supervision.
9. Weak supervision is fast and inexpensive compared to manually creating labels. It can also adapt to changes in the data or requirements. If you have human annotations,

and the requirements change, you need to redo the annotations. If you use weak supervision, you can apply the labeling functions again.

10. Weak supervision has the drawback of producing noisy or inaccurate labels.
11. Active learning is an efficient strategy to label data. It iteratively trains a model and uses it to find which data points are the most valuable to label next.
12. Active learning is especially helpful in scenarios where we have abundant unlabeled data but creating human annotations is expensive or time-consuming.
13. Uncertainty sampling and diversity sampling are the two most common strategies to combine with Active Learning.
14. Uncertainty sampling identifies data points near a decision boundary. These samples have a larger chance of being misclassified by the model.
15. Diversity sampling identifies any unlabeled samples that are unusual, underrepresented, or unknown to the model in its current state.
16. Uncertainty and diversity work best together. Combine both strategies when building a selection function to decide which data points to label next.

Session 2

How To Build A Model (That Works)

Feature engineering

1. Better data is crucial for building better models. The single biggest impact on your model's performance will come from very good features[1].
2. Data cleaning improves model accuracy, generalization, and interpretability. It reduces prediction bias, decreases computational costs, and ensures data consistency and ethical and regulatory compliance.
3. Vectorization is the process of converting data from its original format into tensors or vectors of numbers. Label encoding, one-hot encoding, and target encoding are examples of vectorization.
4. Label encoding replaces each categorical value with a unique integer.
5. Label encoding has the disadvantage of potentially introducing a misleading ordinal relationship among categories, and it's problematic for models that assume a linear relationship.
6. One-hot encoding replaces the original feature with a binary column for each categorical value.
7. One-hot encoding can significantly increase the number of features in the dataset, especially if the categorical variable has many values. It also increases the data's sparsity, and the models might suffer from the Curse of Dimensionality.
8. Target encoding replaces each categorical value with the mean of the target column for that category.
9. Target encoding can inadvertently introduce future information into the model training process, causing overfitting, especially in cases where the number of data points for certain categories is small.

[1] Konrad Banachewicz and Luca Massaron. "**The Kaggle Book. Data analysis and machine learning for competitive data science.**" Interview with Bojan Tunguz, a Kaggle Quadruple Grandmaster. Book link.

10. The effectiveness of target encoding depends heavily on the distribution of the target variable. If the target data changes over time, the encodings might no longer provide a good representation, affecting model stability.

11. Normalization and standardization are techniques that turn numerical data into a consistent range. Models work best when their features have small values with similar ranges.

12. Use normalization when you need a bounded range and the data is not Gaussian (normally distributed). Normalization is particularly helpful in algorithms that compute distances between data points and require scaling to a specific range, like neural networks.

13. Normalization is sensitive to outliers. Outliers can skew the minimum and maximum values, distorting the range for other data points.

14. Use standardization when the feature's data follows a Gaussian distribution or when the algorithm assumes normally distributed data, such as linear or logistic regression.

15. Standardization does not bind values to a specific range, which might be necessary for some algorithms. It also retains the effect of outliers since it merely re-centers and rescales the data.

16. Most models don't work with incomplete data. Handling missing values is a critical step for building reliable, unbiased, and accurate models.

17. You can handle missing values by removing the feature containing the missing values, removing the individual samples, or replacing the missing values with the most frequent feature value, its mean, or median.

18. Be careful when removing missing values. The distribution and frequency of missing values may uncover key insights about your data or process.

19. Feature engineering is the process of transforming and improving the original data into more useful information to train a model. The quality of the features in your training data will have the single biggest impact on the quality of your model.

20. A few examples of feature engineering are binning or bucketing features, deriving additional information from the original data, or creating new features by clustering the existing data.

21. Deep learning didn't kill feature engineering. Feature engineering can enhance a model by providing additional information that raw data doesn't convey.

Choosing a model

1. Before training a model, establish a baseline that will serve as a measuring stick to track your progress. A random baseline and the zero-rule algorithms are examples of simple baselines.
2. A model that can't beat a rudimentary baseline hasn't learned any significant patterns from the data.
3. A random baseline is based on predictions generated using random chance, without any regard to the input data. It gives us a simple and unbiased starting point for comparison.
4. The zero-rule algorithm predicts the most frequent class in a classification problem or the mean or median value in a regression problem. This algorithm is called "zero-rule" because it requires zero knowledge about the data. It bases its predictions solely on the distribution of the target variable.
5. The zero-rule algorithm can be surprisingly effective as a baseline for imbalanced datasets where one class dominates. In such cases, predicting the most frequent class can result in a high accuracy rate.
6. We can also measure people's performance solving the same task or the performance of an existing process and use these values as the baseline.
7. Scope creep will eventually turn any rule-based system into a nightmare. To replace a complex set of heuristics, the best solution is a simple model.
8. Choose the simplest model that could possibly work.
9. The right to introduce complexity must be earned. Additional complexity is only worth it if the extra performance makes a difference.
10. Simplicity is not only about the architecture of the model but about how easy the model is to use.
11. When choosing a model, consider its general capabilities. Consider what your model can and can't do. Understand the patterns it can capture, its sensibility to noise and outliers, and its ability to generalize.
12. When choosing a model, consider hardware, time, and costs. Edge devices have limited resources and require fast inference times. Large models cost more.
13. When choosing a model, consider how the model scales to more data. Your model may work today, but the volume and complexity of data will increase over time.

14. When choosing a model, consider black box versus white box models. This is critical in scenarios where it's important to understand the logic behind a model's predictions.
15. When choosing a model, consider how familiar is your team with it. Prioritize using the models you know best. Lack of expertise is a huge roadblock.
16. Algorithms come and go. The more experiments you run, the better your chances of choosing the best model for your problem.
17. Ignore state-of-the-art models as much as possible. Most of these models are the result of a popularity contest, but they struggle with real-world data.

Tuning and tracking

1. After choosing a model, the next step is building an end-to-end training pipeline. At this stage, we aren't worried about building a good model. We are only worried about writing and wiring the pieces we need to train and evaluate a model.
2. A good strategy when building a training pipeline is to overfit your model to ensure everything works. Train and test your model using a small batch of data. If everything works, your loss should be zero.
3. Instrumentation is key for building reliable systems. Tracking will let us recreate and compare different experiments. More importantly, it will help us understand how different changes affect a model. Track as much as you can to enable debugging and analysis of your machine learning application.
4. After you have an end-to-end training pipeline, focus on building a working model. At this stage, we'll iterate to improve the data and tune the model until we have a good enough solution.
5. Model-centric changes aim to build the best model for a given dataset. Data-centric changes aim to produce the best dataset to feed a given model.
6. A few examples of model-centric changes are hyperparameter tuning, implementing ensemble models, using custom loss functions, transfer learning, using regularization techniques, and modifying the model architecture.
7. A few examples of data-centric changes are feature engineering, improving labels, data augmentation, balancing the dataset, generating synthetic data, and collecting additional data.

Distributed training

1. Distributed training helps build better models. It allows us to use more computing power at once. Distributed training reduces training time and helps us handle large datasets and models.
2. Data parallelism and model parallelism (with pipeline execution) are the two distributed training techniques used in production applications.
3. Data parallelism replicates the same model across multiple nodes and trains each replica on a different subset of the data.
4. Model parallelism splits the model across different nodes and trains each portion using all of the data. Model parallelism only works when we can split a model into independent components that we can run in parallel.
5. Pipeline execution is a technique to perform model parallelism while minimizing idle nodes.
6. Data parallelism is easier to implement and it works with every type of model. It has the downside that each processor needs to hold the entire model in memory, limiting the size of the model that can be trained.
7. Model parallelism is more complex to implement as it requires splitting the model and managing dependencies between different parts. It introduces a significant communication overhead.

How To Evaluate A Model

Evaluation strategies

1. Evaluation is critical for building working models. A good evaluation strategy is the difference between a model that works and a waste of electricity.
2. You may have a model that performs well with your small dataset, but without a solid evaluation strategy, it may not work with real production data.
3. Understanding the context of evaluation metrics is essential before interpreting them. You can use a random baseline or the zero-rule algorithm. You can also compare your model to existing solutions and human benchmarks.
4. Technical metrics don't represent business interests. Collaborate with business experts to create business-relevant metrics. This step is crucial to ensure your model solves the right problem.
5. The easiest way to evaluate a model is to test it on a holdout set, a small sample of the data the model has never seen before. Use this holdout set to infer the performance of the model on future, unseen data.
6. A holdout set may lead to a biased model assessment. It's also an inefficient way to take advantage of the data, especially when working with small datasets.
7. Cross-validation provides a better way to assess the quality of a model. It leads to a less biased estimate of the model's performance and it's ideal for smaller datasets.
8. Cross-validation splits the available data into k folds, and for k iterations, evaluates the model using one of the k partitions and trains with the rest of the data.
9. After using cross-validation, you can train a new model using the entire dataset. This is a great strategy that works best when training is cheap and data is valuable but scarce.
10. After using cross-validation, you can use every model generated during each iteration and average their results. This is a good strategy when training a new

separate model is too costly or impossible because you used a different process to train each individual model.

11. Cross-validation is prohibitive for large datasets. It is too costly because it involves multiple rounds of training and validation of multiple models.

Preventing data leakages

1. Data leakages lead to useless models.
2. A data leakage happens when information related to the ground truth is introduced within the training data.
3. Data leakages are one of the most common issues when building machine learning models.
4. A leaky validation strategy happens when information from the training data "leaks" into the validation data when building a predictive model.
5. All feedback channels are model training mechanisms. To prevent overfitting, always set your data aside without looking at it. Fitting operations on the test data is a common way to introduce leakages.
6. A common misconception about overfitting is that you can only overfit a dataset if you are directly training on it. In reality, you can overfit to anything, as long as you have a feedback loop connecting the results on the target dataset and your model[2].
7. A plane's life span isn't measured in years but rather in pressurization cycles. Every time a plane takes flight, it is pressurized, which puts stress on the fuselage and the wings. Conversely, we can measure the quality of a test set in "tuning cycles."Every time we use the test set in any way, we put pressure on it and overfit our model.

Going beyond the test set

1. Prediction fairness helps prevent discrimination at scale. Humans can discriminate, but an unfair model can do it on a massive scale.
2. Fairness is one of the most important characteristics of models that impact people.

[2] François Chollet. Feb 13, 2024. https://twitter.com/fchollet/status/1757573833677308132.

3. To evaluate the model's fairness you can use invariance tests. You can also introduce controlled changes to the model's inputs to ensure they lead to predictable outputs.
4. Summarization metrics conceal how models work. Reducing evaluation to a single metric is dangerous. It may hide severe failure cases for slices of the data. For example, reporting the average evaluation score over many samples may under-represent severe failure cases for rare samples or subpopulations in the data.
5. Simpson’s Paradox is a phenomenon in which a trend appears in separate subsets but disappears or reverses when we combine all the data.
6. Identify and split your data into relevant subsets. Evaluate your model's performance on each.
7. Backtesting evaluates a model on historical data. It uses historical data to simulate how a model would have performed in the past. Backtesting ensures that a model not only fits the data it was trained on but also holds up against unseen or future scenarios.
8. Look for disproportionately important samples. They are often critical for the business, but global metrics hide how models perform on them. For example, some of the customers of the model might be big spenders compared with the rest. These power users are disproportionately important.
9. Error analysis helps identify underperforming slices. It also finds data subpopulations that share common characteristics and could improve the model.
10. A simple strategy is to look at the predictions the model gets wrong, and apply a clustering algorithm to find any common characteristics among them. You can focus on these results to improve the data and the model results.
11. A step-by-step example of how to apply error analysis is to start by identifying slices that might be relevant to improving the model. Using the characteristics of these slices, label the dataset, compute the model’s performance on each of the labels, establish a comparison baseline to understand how big is the error gap between the model’s results and the baseline, determine how frequent is each label in the dataset, and compute the potential impact of fixing one of the slices.

Working with imbalanced data

1. Knowing how to build and evaluate models trained on imbalanced data is crucial in real-world applications.
2. We usually focus on identifying the minority class. For example, identifying credit card fraud, rare diseases, spam filtering, network attacks, manufacturing defects, or any other anomalies.
3. Undersampling reduces the size of the majority class to balance the dataset. This is done by randomly removing instances from the majority class until the class distribution is more balanced.
4. Undersampling may cause information loss by removing data points that represent valuable information for building a robust model.
5. Oversampling increases the size of the minority class by duplicating existing instances or generating synthetic instances, thus balancing the class distribution.
6. Oversampling can lead to overfitting because it involves duplicating data points or creating very similar ones. It can also increase computational costs because it creates additional data points.
7. Class imbalances aren't a problem in themselves. Correcting imbalances using resampling techniques may worsen model performance[3].
8. Using SMOTE to correct class imbalances does not improve prediction performance when using strong classifiers[4].
9. You can handle imbalances by using the appropriate metric for your problem, more robust algorithms, changing the class weights, cost-sensitive learning, or threshold moving.
10. When using threshold moving, don't use fixed thresholds. Instead, a more robust strategy is to learn the appropriate thresholds by evaluating using the holdout validation data.

[3] Ruben van den Goorbergh, Maarten van Smeden, Dirk Timmerman, Ben Van Calster. "**The harm of class imbalance corrections for risk prediction models: illustration and simulation using logistic regression.**" Journal of the American Medical Informatics Association, Volume 29, Issue 9, September 2022, Pages 1525–1534, https://doi.org/10.1093/jamia/ocac093.

[4] Yotam Elor, Hadar Averbuch-Elor. "**To SMOTE, or not to SMOTE?**" arXiv:2201.08528v3.

11. Data augmentation exposes a model to more aspects of the data, making it more robust to noise and improving its ability to generalize to unseen samples.
12. Some common data augmentation techniques in computer vision are rotation, flipping, cropping, and scaling.
13. MixUp[5] is a technique that helps deep neural networks avoid memorization and sensitivity to adversarial examples. It's useful to augment a dataset of images by blending existing samples and turning discrete labels into continuous labels.
14. A clever technique for augmenting text data is to use back translations, which automatically translate text back and forth between different languages to generate alternatives.
15. A popular technique to generate large amounts of synthetic data is to use a template and automatically fill in the blanks using a dictionary of words.

[5] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, David Lopez-Paz. "**mixup: Beyond Empirical Risk Minimization.**" arXiv:1710.09412v2.

Session 4

One Hundred Ways To Deploy A Model

Model versioning

1. Building models is not only about writing code. We need a different process to store, version, share, maintain, and serve model predictions at scale.
2. Model versioning is a crucial part of the machine learning workflow. It helps track and manage changes to source code, data, metrics, parameters, and hyperparameters.
3. The model registry serves as a central versioning hub. From here, we can export a specific version of the model and deploy it to start serving predictions.
4. The Model Registry can store a model's parameters, metadata, version information, metrics, artifacts, and lineage.
5. Versioning models allow us to reproduce a model, maintain the model's history, compare different model versions, and share artifacts among different teams.

Serving strategies

1. We can deploy and serve the predictions of a model by wrapping it with a RESTful API, using TensorFlow Serving, TorchServe, or Nvidia Triton.
2. We can host a model on the cloud for scalability. We can also host a model on-premises when working with sensitive data that can't leave local infrastructure.
3. We all want to serve good predictions fast and cheaply. Quality, speed, and costs are the key tradeoffs to consider when serving model predictions.
4. The unattainable priority triangle consists of quality, cost, and speed. You can only achieve two of these at the same time. For example, you can have a cheap and fast model with low quality, a good and cheap model that's slow, or a good and fast model that's expensive.

5. Latency is a measure of the delay experienced by a system. Lower latency indicates a more responsive system, which is crucial for real-time predictions.
6. Throughput refers to the amount of data processed by a system in a given amount of time. High throughput is crucial for serving many predictions quickly.
7. Depending on the system, we can generate predictions on demand (dynamic serving) or pre-generate them and serve them from a cache (static serving).
8. Dynamic serving computes predictions in real-time. It has lower storage costs and variable latency. It works best for problems with a large potential number of input values (high cardinality). An Example of a high-cardinality problem is luxury bag pictures taken by users with their cell phones.
9. Dynamic serving it's inefficient when processing large volumes of data in terms of latency and costs.
10. Static serving pre-computes predictions ahead of time. It has higher storage costs and low latency. It works best for problems with a small potential number of input values (low cardinality). An Example of a low-cardinality problem is the potential price of a luxury bag given its brand, model, and condition.
11. Static serving is more efficient for processing a lot of data and reduces latency, but it requires knowing which predictions to generate in advance. It makes the system less responsive to changes.
12. A hybrid system that uses static and dynamic serving is often the best approach. Pre-compute and serve low-cardinality predictions from a cache, and serve other predictions in real time.
13. Deploying models on edge devices requires creativity. We can run large, complex models by breaking a problem down and using two-phase predictions.
14. In a two-phase prediction scenario, we deploy the smaller, cheaper model on the edge and deploy the larger, more expensive model on the cloud. Trigger a request to the large model only when needed.
15. An example of a two-phase prediction setup is building a small, always-running model to listen for an activation phrase. We use this small model to trigger requests to a second, more complex model that runs in the cloud and understands any user query.
16. Quality is inversely proportional to complexity. Consider enforcing modularity by decomposing complex problems into separate modules. Break down

complex problems into smaller tasks and build separate models that focus on different aspects of the problem.

17. An inference pipeline is a sequence of models working together to make a prediction. In a pipeline, the output of one model becomes the input of the next.
18. Evaluate and monitor each pipeline model individually. A pipeline amplifies errors and hides each model's individual strengths and weaknesses.
19. A technique for evaluating the individual potential of every model in a pipeline is to compute the performance of the end-to-end pipeline and iteratively replace each individual model with perfect predictions before reevaluating the performance. We can then focus on the model with the potential to improve the pipeline the most.

Human-in-the-loop

1. Implementing human-in-the-loop systems is one of the most powerful ways to design applications that augment human judgment for critical decisions.
2. A human-in-the-loop architecture improves a system's trustworthiness. However, its downside is its complexity and the need for additional resources.
3. Using probability thresholds or uncertainty can help decide whether the system should involve a human instead of taking direct action.
4. A cost-sensitive architecture will allow the system to compute the potential cost of acting on a prediction and compare it with the cost of using human help. Combining cost-sensitivity with human-in-the-loop is one of the most powerful ways to build machine learning systems.
5. We can also determine whether to involve a human based on previous experiences handling data that's similar to the current request. For example, we can find similar previous requests and determine how well the model handles them. If the performance is good enough, we can let the system handle the request. If not, we can involve a human.
6. Test-time augmentation is a good trick for improving your model predictions during inference time. It involves augmenting requests and averaging the model results.
7. With test-time augmentation, you augment the request before running it through the model and then average the prediction results. For example, instead of running a picture through the model, you can augment the original

image to generate two additional versions, make predictions for all of them, and average the three softmax vectors you get back to determine the final result.

8. The success of test-time augmentation depends on how good your augmented samples are. If you create sloppy variations of the original image, test-time augmentation can quickly decrease your model's performance. Keep it simple. Success relies on avoiding excessive complexity.

Model compression

9. Model compression reduces a model's size without significantly sacrificing its accuracy. Smaller models take less space and run faster. The three most common model compression techniques used in practice are Pruning, Model quantization, and Knowledge distillation.
10. Pruning is a model compression technique that identifies the parameters that have no or negligible influence on the final predictions and sets them to zero.
11. Pruning increases the sparsity of the model making it faster. An increased scarcity makes models more computationally efficient and reduces the space we need to store them.
12. Pruning increases the risk of a model to overfit. Finding which parameters to prune and how much pruning is enough is a difficult and expensive process.
13. Quantization reduces the precision of the model parameters by using fewer bits to represent them. This reduces the model size and increases its speed.
14. Quantized models take less space and run faster, but they can represent a smaller range of values, leading to more rounding errors and worse performance.
15. Post-training quantization is the most popular and easiest way to compress models. Quantization-aware training is better for accuracy preservation.
16. Knowledge distillation is another compression technique that transfers the knowledge of a large model—the teacher—into a smaller model—the student.
17. Knowledge distillation can significantly reduce the size and computational complexity of a model.
18. In Knowledge distillation, Student models often outperform models of similar size trained without distillation, as they learn from the refined knowledge of

the teacher model. The process may lead to a loss in depth of understanding or model capabilities.

Session 5

How To Monitor A Model (Drift Is A Bitch)

Edge cases and feedback loops

1. Building models is easy. The challenge is to keep them alive.
2. Models degrade over time. Every model has to deal with edge cases, positive feedback loops, and data distribution shifts.
3. An edge case is an extreme sample that causes a model to make outrageous or catastrophic predictions. Identifying outliers is a mechanism to prevent them.
4. Some example techniques to identify outliers are isolation forests, computing the distance between samples, and reconstruction-based methods like autoencoders.
5. Edge cases aren't the same as outliers. An outlier might be an edge case, but not all outliers are edge cases. An edge case refers to an example where the model performs significantly worse. An outlier refers to an example that's significantly different from the rest of the data, but the model might handle it well.
6. Edge cases are inevitable. They can occur because of damaged sensors, malicious input, bad data collection, corrupt data, or rare events.
7. Feedback loops are a common problem in production. They cause systems to influence their own behavior and make it difficult to analyze model performance.
8. An example of a positive feedback loop is building a model to recommend the top most popular songs. As customers listen to the recommendations, the songs in the list will become even more popular.
9. A hidden feedback loop occurs when feedback is many steps removed or occurs indirectly. These loops are much more difficult to detect.
10. An example of a hidden feedback loop is a company predicting the list of clients that might churn at the end of the month. The sales team uses this information to offer an incentive to those clients, causing some of them to stay.

This information will bias the feedback that goes back to the model. You will never know if the original prediction was correct.

11. Remove hidden feedback loops whenever possible. Add variability using exploration/exploitation or randomization, or add positional features to the data.

Distribution shifts

1. You can't out-train bad data. If your training and production data come from different distributions, your model will suck.
2. Adversarial validation is a clever, simple technique that will help you determine whether two datasets come from the same distribution.
3. To run adversarial validation, join your train and test set and replace the target column with a new binary feature. Set the value of this feature to 0 for every sample from your train set and 1 for every sample from the test set. Train a simple binary classification model on this new dataset. If you can tell train from test apart, your data comes from different distributions.
4. Data distribution shifts happen when production data diverges from the training data. They are the biggest challenge any production model faces and degrade its performance over time.
5. Concept drift occurs whenever the relationship between model inputs and outputs changes because the model's underlying assumptions aren't the same.
6. An example of concept drift is how credit card fraud has changed over time as providers figure out ways to prevent illegitimate transactions. In 2009, we had magnetic bands that were easy to hack. Most of the fraud happened physically at the point of sale. In 2024, we have security chips, so most of the fraud happens online.
7. Data drift occurs whenever the input data used by the model to make predictions changes as compared to the data used to train the model.
8. An example of data drift is when a model that you trained on data from young users starts processing data coming from older people.
9. Data drift can happen because of bugs, changes in the input data schema, changes in the distribution of a feature, or the meaning of the data changes over time.

10. We can experience drift suddenly, when a new concept occurs within a short time, gradually, when a new concept gradually replaces an old concept, incrementally, when an old concept incrementally changes to a new concept, or on a recurrent basis, when an old concept reoccurs after some time.

Monitoring strategies

1. Unit testing of individual components and end-to-end system testing are important but not sufficient. The best way to identify model degradation is to continuously monitor your model's performance over time.
2. Monitoring refers to the act of tracking, measuring, and logging different metrics to determine when something goes wrong. Monitoring helps maintain model reliability over time.
3. We can monitor model inputs, operational metrics, predictions, and user feedback generated by the model.
4. Any changes in the input data used by a model are useful signals for understanding the system's health. Monitoring should validate the schema of input data.
5. Track that input features are within acceptable ranges, their statistics, and categorical values belong to a predefined set and follow the correct format. For example, compute the minimum, maximum, mean, and median values of a feature and ensure they are within an acceptable range, or ensure that the value of one feature is correct in comparison to the value of another feature.
6. Operational metrics convey the health of a system. It doesn't matter how good a model is if you can't keep the system up and running serving its predictions.
7. Some operational metrics you can monitor are latency, throughput, hardware utilization, number of requests, and how many of them were successful. For example, you can monitor the percentage of CPU and GPU usage and the memory utilization of a deployed model.
8. Monitoring should also validate the distribution of model predictions and track how specific model metrics—e.g. accuracy, precision—vary over time.
9. Assuming your model hasn't changed, any changes in the distribution of the model predictions generally indicate a distribution shift in the inputs.
10. Ground truth data is essential for model monitoring. The system should store and label model requests to determine how model performance changes over time.

11. Monitoring changes in user feedback for your model predictions is another indication we can use to detect changes in the performance of the model.

Keeping models alive

1. Finding alternatives to volatile features will help design models that are more resistant to drift. For example, bucketize features that change too much. Transforming a stock's trading volume into buckets to slow down its rate of change is an example of using a more stable feature.
2. Isolate models susceptible to drift. Create separate models to handle volatile data and prevent them from causing drift for the entire system. For example, instead of using a single model to predict real estate prices in Florida, we should build one model to predict prices in Miami, and another model to predict prices in the rest of the state.
3. The more data you use to train a model, the more likely the distribution of production data will match the distribution of the training data.
4. If possible, collect more data to build models more resistant to data distribution shifts. Additional data decreases the chance of data drift. Unfortunately, we don't have the time, money, skills, or hardware to collect the perfect dataset.
5. Retraining is the best strategy to combat drift. A combination of continuous updates and evaluation is the key to adapting to data distribution shifts.

Session 6

How To Build A Continual Learning System

Continual improvements

1. Real-world applications don't work with static data.Production machine learning systems must be capable of adapting and learning from new data.
2. The machine learning cycle starts with data collection, labeling, training a model, and evaluating it. If the model is good enough, we deploy it and monitor it. Whenever the model's performance dips, we collect more data and repeat the cycle.
3. Most people retrain models whenever they feel like it.Continual learning is the process of automatically updating models in a production system.
4. Never ask how frequently you should retrain a model.Instead, start retraining as frequently as you can.Make it work, make it right, make it fast.
5. Periodic retraining is suboptimal, but it's a good baseline strategy. Other triggers include retraining based on the volume of data or the amount of drift.
6. Retraining as frequently as you want is expensive.Consider the trade-off of how much performance degradation is acceptable in relation to the cost.

12. Determine the importance of fresh data to understand how frequently retraining should happen. Fresh data isn't always worth it.
13. By comparing the performance of different models trained with data of varying recency, we can estimate the rate at which a model's performance falls off over time and how often it might be necessary to retrain it.
14. An important aspect to focus on when building continual learning is deciding what data and how much of it you should use to retrain the model. Using every available data point to retrain a model doesn't necessarily lead to better results. Data is perishable.
15. Data is only valuable if it increases diversity and improves the representation of real-world scenarios.Trim anything that doesn't lead to better predictions.

16. We can estimate the importance of historical data by comparing the performance of different models trained with data covering different periods of time.
17. Labels are fundamental for continual learning.Labeling will likely become the bottleneck that will limit the retraining frequency of your models.

Retraining strategies

1. The most popular option to retrain a model is doing it from scratch using all of the data you have available.This strategy is also known as stateless training.
2. You can also retrain your model incrementally by continuously updating it using new data.This strategy is also known as stateful training.
3. Training from scratch is slow and expensive.As learning frequency increases, stateless training becomes less viable. The advantage of training from scratch is its simplicity: you already have it working!
4. GrubHub went from daily stateless retraining to daily stateful retraining reducing their training compute costs 45 times[6].
5. Stateful training is faster and cheaper because we only need to use new data to retrain the model.Its main disadvantage is the potential for the model to suffer from catastrophic forgetting.
6. Catastrophic forgetting is the tendency of models to completely and abruptly forget information they previously learned upon learning new information.
7. The best approach is a hybrid training strategy. Implement incremental training as the main ingredient and combine it with periodical stateless training sessions.

Testing in production

1. Offline evaluation is not enough to test a model.Continual learning requires a different testing strategy to ensure continuous improvements.
2. Offline evaluation cannot determine how a model will react to production data it has never seen before.We need to test the model in production.

[6] Alex Egg. "**Online Learning for Recommendations at Grubhub.**" arXiv:2107.07106.

3. A/B testing is a straightforward testing strategy.It limits the potential impact of mistakes and provides a clear and direct comparison between models.
4. A/B testing exposes some users to a potentially inferior model. It requires a significant amount of traffic and time to reach statistical significance.
5. Canary releases are great for ensuring the operational stability and reliability of the entire system.We can control who is exposed to the candidate model.
6. A canary release will also expose some users to a potentially inferior model. The feedback we collect from these users may not represent the larger user base.
7. Shadow deployments allow us to test a new model under real-world conditions without exposing any user to a potentially inferior model.
8. Shadow deployments increase the load on infrastructure resources. We can't test user interactions resulting from model predictions.
9. Interleaving experiments allow for a direct comparison of models under identical conditions.It provides immediate feedback about user preferences.
10. Setting up interleaving experiments is challenging. It requires a sophisticated process to correctly attribute user preferences and actions to individual models.