Machine Learning Foundations Unit 8: Prepare ML Models for the Real World

Table of Contents

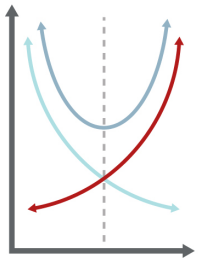
- Quiz: Check Your Knowledge: Feature Issues
- Assignment: Unit 8 Assignment - Choose Your Machine Learning Problem and Data
- Assignment: Unit 8 Assignment - Written Submission
- Module Wrap-up: Addressing Feature Issues
- Ask the Expert: Mehrnoosh Sameki on Summarizing the Machine Learning Life Cycle
- Assignment: Lab 8 Assignment

Unit 8 Overview
Video Transcript

Machine learning engineers are generally accountable for the stability and performance of their model systems, where performance here can be defined in multiple ways. There are several ways in which ML systems can cause problems and it is important for ML engineers to understand and learn how to debug such problems. In this series we introduce three classes of model failures that we call performance failures, execution bottlenecks, and societal failures. Becoming great at avoiding and mitigating the various failure models will require a deeper study, as well as good old fashioned experience. But now is still the best time to start learning and practicing this discipline. After introducing various causes of the different failure models, we'll cover common debugging techniques and show how following common model development best practices is often the best way to start building reliable modeling systems.

What you'll do:

- **Follow software development best practices**
- **Explore common ML failure modes**
- **Diagnose how data size contributes to execution bottlenecks**
- **Diagnose how feature issues contribute to degraded model performance**
- **Create and implement a project plan to solve a machine learning problem**



Unit Description

When solving real-world ML problems, machine learning engineers must think about the data they are working with and how to produce a “good” model that will perform well in a production environment. There are many factors to consider on top of preparing data and following good software engineering techniques. In this unit, Mr. D'Alessandro introduces three types of failure modes: model performance,

execution bottleneck, and societal failure. We will focus on the first two failure modes; in the next unit, we will discuss how to address societal failure using ethical AI.

At the end of this unit, you will practice proper problem formulation by choosing your own data to work with and preparing it to solve a specific ML problem while mitigating bias. The three data sets you may choose from are all data sets you've seen and worked with in previous weeks. You will formulate your problem and draft a project plan that describes your model fitting, evaluation, and improvement decisions.

Back to Table of Contents

Tool: Unit 8 Glossary

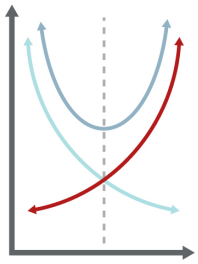
Though most of the new terms you will learn about throughout the unit are defined and described in context, there are a few vital terms that are likely new to you but undefined in the course videos. While you won't need to know these terms until later in the unit, it can be helpful to glance at them briefly now. Click on the link to the right to view and download the new terms for this unit.

Download the Tool

Use this **Unit 8 Glossary** as a reference as you work through the unit and encounter unfamiliar terms.

Back to Table of Contents

Module Introduction: Explore Solution Engineering



"Solution engineering" is a term that Mr. D'Alessandro uses to encompass the idea of mitigating errors while developing projects. You will gain experience following model developer best practices as you continue your journey to become a machine learning engineer. This includes making good design decisions and using various techniques to improve your implementations.

In this module, Mr. D'Alessandro lists the best ways to help effectively reduce machine learning-based risk. He also explains how to avoid common machine learning failure modes. These are all things to keep in mind when you begin professional machine learning projects.

Back to Table of Contents

Watch: Introduction to Machine Learning Failure Modes

Since machine learning models are driven by software, we have to consider common software issues as well as failure modes specific to machine learning. In this video, Mr. D'Alessandro introduces three classes of failure modes.

Video Transcript

Building effective machine learning models takes a lot of care — from defining the problem, to preparing the data, and finally training and validating different model candidates. There are plenty of places where the process can break down. Machine learning models are driven by software. Everything we do in this field requires us to consider common issues associated with software engineering, such as bugs in code, deployment problems and scalability, or security failures. On top of common software issues, we have to consider failure modes that are specific to machine learning. When I say failure mode here, I'm referring to a particular way in which a system can fail. For instance, in software systems, two types of failure modes are unaccounted-for edge cases, and a system that is unable to accommodate a sudden burst of traffic. Machine learning has its own specific set of failure modes. The best failure modes, in my opinion, are those that completely break your system. While certainly frustrating and often expensive, these are clear failures that will be identified immediately, and ideally remediated quickly. Machine learning has these failure modes, of course, but these tend to be caused by common software issues, such as having code errors, or having issues with APIs and data types. Machine learning models also have many silent failure modes. These are typically failures specific to machine learning concepts. Very often, the modeling and prediction processes will run smoothly, though the outputs aren't what we had intended. I'm going to present three classes of failure modes and give high level descriptions of each. I will present strategies for addressing each failure mode in separate models and videos. The first class of failure mode is a model performance failure. A performance failure can be described in two ways. When evaluating on a test set, your model may have poor performance or your model's performance might be too good to be true. The latter case may seem strange at first, but this is one that can lead to overconfidence in a system, which is a bad place to be, in my opinion. Both of these lead to poor generalization performance when the model

is applied to real life data. The second class of failure is what I call an execution bottleneck. This is a common and frustrating issue, which I can describe succinctly as a data preparation or modeling process not terminating in a reasonable amount of time. This is commonly caused by a misalignment between the data you have and the tools and hardware you're using to process the data. The third class of failure is what I call a societal failure. This is the case when a machine learning model produces unintended discrimination or disparate impact and generally lacks accountability. I use the phrase societal failure because this is where the failure exists between the owner of the model and the social context in which the model operates. It could be that models with societal failures perform well and generate a lot of value for the model owner. As a result, there are often misaligned incentives in remediating such failures. Progress in this area will come from increased governmental regulation of AI systems and proactive remediation by machine learning engineers. In short, adopting methods to avoid societal failures, which we'll call fair AI, being mindful of machine learning best practices early on in your model development is one way you can contribute to reducing this type of failure mode.

Back to Table of Contents

Watch: Using Good Problem Formation

We can use good problem formation to avoid common machine learning failure modes. Let's take a look at some common design decisions you'll make on professional machine learning projects and where to start when preparing your design decisions.

Video Transcript

Avoiding common machine learning failure modes usually starts with good problem formation. I hope it is clear by now that as a machine learning engineer, there are a lot of subjective choices you have to make. I'll call these subjective choices design decisions, as I think the word design makes it clear that these are not usually empirical choices. Thoroughly understanding the constraints in which you operate, and getting peer alignment ahead of doing any development is one way to proactively avoid the common machine learning failure modes. Let's start by presenting some common constraints to consider. Usually this is the first place to start when preparing your design decisions. Constraints should help you understand what you can and cannot do. Naturally, this is something to consider upfront. The first two constraints are centered around data. It is important to ask yourself whether you have the right data given specific problem statement. When considering whether the data is right for your application, you should look at whether your label captures the spirit of the problem statement, whether you have access to features that might predict the label, and whether you have access to a representative population. Next, we would consider data size. You can have both too much or too little data. Knowing this in advance can help you determine which modeling algorithms would be acceptable to test and what kind of computing systems you should set up. Next, it is important to ask yourself, do you understand a particular algorithm you are planning to use? Building any model can be pretty easy, but building a good model requires you to really understand nuances of the underlying model, and in particular, the art of managing the hyperparameters. Don't be afraid to ever rule out certain modeling algorithms as candidates when you're not confident in how they work. Next, you should work through the scalability requirements of your system. This is usually centered around deployment, but some systems need to make predictions in less

than a second. Scalability requirements might dictate what features and how many are reasonable to deploy and which modeling algorithms are appropriate. The last two constraints here are centered on societal accountability, which involves regulations and interpretability. It is important to understand up front whether there are any regulatory or interpretability requirements that need to be considered. These requirements often dictate what is an acceptable label, what are acceptable features, and what types of modeling algorithm should be used. After documenting your constraints, you will move into the design phase. Here is a list of common design decisions you'll make on almost every professional machine learning project. I've put them into a table here for easy review, and I've mapped out the specific risk categories that are often associated with these design decisions. The first here is translating your problem goal to a label. Remember that we often rely on observable proxies to serve as labels. If the proxy isn't correlated with a real problem, this can lead ultimately to performance and societal failures. One example of this is using data scraped from a resume to predict if a candidate would be a good employee. In this case, the proxy used as a label would be whether or not previously hired employees reached a certain tenure. This could bias the model to only prefer candidates with certain genders or ethnic backgrounds because of underlying biases in the training data. We'll cover this point more in a later lecture. The choice of tech stack should have clear relationships with execution bottlenecks. When I say tech stack here, I am referring to the choice of computational resources and tools used to prepare the data, train the model, and deploy it. If you don't have sufficient computational resources, then you will be less likely to efficiently search through an appropriate set of model candidates, so you might end up with a less performant model from a prediction perspective. The next two, sample size and features used, have implications to all risk categories. In later modules, we'll go deeper into these risks in common mitigations. The last two involve the model selection and evaluation stages. Even though these are later stages in the process, we should always specify in advance our intentions. In particular, we should be clear on how we will evaluate, specifying which metrics are appropriate and which segments of the data we plan to evaluate. Oftentimes, societal failures are caused by poor planning. By this, I mean neglecting to even evaluate your model against common fairness metrics.

Back to Table of Contents

Watch: Consider Model Developer Best Practices

Model developer best practices can help you to reduce machine learning-based risk effectively. In this video, we will cover four best practices including Agile model development, applying unit tests, writing code to be reproducible, and creating good documentation.

Video Transcript

Knowing that good machine learning starts with you, the machine learning engineer, I'd like to introduce model developer best practices that should help you effectively reduce machine learning based risk. In this video, I'm going to present four best practices and how they can help you. The first is always practice Agile model development. Agile model development is the process of increasing your model complexity in separate efforts, as is illustrated by this curve. Your first model should be intentionally simple. This could mean using only a few simple features and a model algorithm like logistic regression that doesn't require much tuning. Over time, you will learn more about your problem, and the feedback should help you build more complex versions that are stable and lower in various risks. The next best practice is to always apply unit tests where applicable. Unit testing is a software development process in which the smallest testable parts of an application called units are individually and independently tested for proper operation. I think this step is particularly applicable for data processing stages. Some easy unit tests on data pipelines can be: test that data types match prescribed data schema, make sure that all values of a feature are within acceptable ranges, and assert that data transformations yield the right row or column count. The next best practice is to write your code and processes to be reproducible, this practice is both practical, and it increases the accountability of your work. It is pretty common to find errors in your work that require you to run through a set of steps again. This might be caused, for instance, by finding upstream issues in your data. On the practical side, having reproducible data and model building pipelines will make any rework necessarily seamless and fast. On the accountability side, reproducibility is a kind of insurance against future issues. If at any point you encounter performance or regulatory problems, being able to reproduce a model can help you debug exactly what is

causing these problems. Writing reproducible pipelines takes upfront planning and work. If you are rushing to finish a project, designing the right code abstractions and functions will slow down development time. It is ultimately up to you to use your judgment and decide how to balance the trade-off between execution speed and reproducibility. Finally, related to reproducibility is creating good documentation. The documentation starts in the planning phases, where ideally you'll create a strong project brief that outlines the motivation from building a model and the methods and data you will use to build it. Following that, having easy to read and documented code will help with peer review and collaboration. Finally, someone should be reviewing your model's output and performance metrics. Creating written documentation that clearly reports your model strengths and limitations is a great way to ensure that what you might ultimately deploy is doing what it is intended to do. These four practices take proactive planning and increase upfront development costs. Again, it is up to you to balance the trade-off between execution speed and the overall system reliability and risk. For models that ultimately make it into your production systems, proactively managing risks is usually always worth it.

Back to Table of Contents

Quiz: Check Your Knowledge: Solution Engineering

You may take this quiz up to three times.

The full contents of this page cannot be rendered in the course transcript. Please complete this activity in the course.

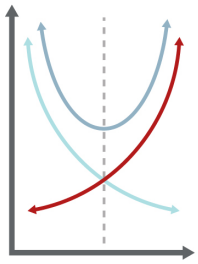
Back to Table of Contents

Module Wrap-up: Explore Solution Engineering

In this module, Mr. D'Alessandro listed the four best practices used to reduce machine learning-based risk effectively. These include Agile model development, applying unit tests, writing code to be reproducible, and creating good documentation. Mr. D'Alessandro also described various ways of avoiding common machine learning failure modes. You will find that this becomes easier over time with experience as you work on more and more machine learning projects. Although choosing good data is important, remember to follow model developer best practices as you prepare for future machine learning projects.

Back to Table of Contents

Module Introduction: Considerations for Data Sufficiency



How much data is needed? Is the data set too large? Is the data set too small? These are some questions machine learning engineers may ponder when preparing to solve machine learning problems. In this module, Mr. D'Alessandro will introduce other considerations when it comes to data such as bottlenecks, bias-variance tradeoff, and class imbalance. Mr. D'Alessandro will also introduce a way of measuring data size efficiency.

Back to Table of Contents

Watch: Managing Execution Bottlenecks

Here, we will take an in-depth look at execution bottlenecks, one of the failure modes introduced earlier. Mr. D'Alessandro defines an execution bottleneck as a data or modeling process that is not terminating in a practical amount of time.

Video Transcript

One of the failure mode classes I introduced earlier was execution bottlenecks. I define an execution bottleneck as a data or modeling process that is not terminating in a practical amount of time. Of course, the operative word here is practical. Machine learning pipelines can take 10 minutes or days to finish. Deciding what is an acceptable run time limit is up to you and your general problem needs. Out of all the failure modes, this is the one that will likely frustrate you the most when doing practice problems as a student. So, managing this risk generally starts now. The primary pattern that drives execution bottlenecks is too much data relative to your computational resources. This simple pattern description immediately yields two effective yet simple solutions. The first solution, and the one most accessible to a student, is to reduce your data size. We can reduce data size in both data preprocessing steps by either downsampling or using batch processing. Downsampling is the process of taking random subsets of your data and running the full lifecycle on the sample data. When I am in the process of developing and testing code, I typically always use a sample of data. This helps me avoid runtime bottlenecks, particularly during testing. When I am satisfied with the code, I'll run it end to end on the full data set. There is a natural performance tradeoff, though. Using smaller data sets might reduce your performance potential, so this is something to be aware of; and by performance potential, I mean predictive performance. The next strategy is to use batch processing. Batch processing is where we read partitions of the data into memory and perform data operations on each partition, one at a time. This is pretty effective for most data preparation steps. I should note that not all machine learning algorithms accommodate batch processing during the training step. This is normally available in logistic regression or neural networks where we are using what are called gradient optimization techniques to fit model weights that minimize the loss function. Whether you choose to use batch processing or downsampling, this is often

the best way to deal with large data sets when you're working on a personal computer and are hitting execution bottlenecks. The next strategy is to keep all the data you have, but add more computing power. This is usually very effective but is more likely an option in a professional setting than at home. Computing power costs money, so you're ultimately trading off execution speed for monetary cost. The last strategy aligns with a 'work smarter, not harder' philosophy. This strategy is to parallelize your processing as much as possible. This is probably the best option out of all that I've mentioned, but it takes some specialized knowledge and computing environments to get it right. I want to describe parallel processing at a high level and then walk through a particular example. The diagram here shows data that is going through parallel processing. First, we assume we have a set of independent computer processors. These could be separate servers or processors within a given computer. The data gets either replicated or split to each processor, and then each processor performs some compute logic on the associated data. After each processor has finished its operation, some desired output is sent to a central server for aggregation and further processing. There is a framework called MapReduce that is often used to describe this whole process. The map step is where data and code are sent to be processed in parallel. The reduce step is where the aggregation of the parallelized results take place. Many data and modeling processing algorithms are naturally parallelizable. Most modern database engines will use a MapReduce framework under the hood to run data processing pipelines. The parallelization is usually done for use. All you would have to do is run basic SQL commands. Additionally, models like decision trees and random forests are easy to parallelize. So in short, parallel processing is appropriate where computing on one data partition doesn't depend on the results of computations on other data partitions. Cross-validation is one such process. Let's say we are running a threefold cross-validation. We can show how this would work in a parallel computing environment here. This is the same diagram as before, but I've specified exactly what the compute processes are doing and we're using a replica of the data instead of a split of the data in each processor. Each processor is training and validating a model on the appropriate folds. The results would then be passed to the aggregator for model selection. If you have K processors working in parallel, you can perform K-fold cross-validation as fast as it would usually take to build a single model. This whole discussion on parallel processing was a bit on the conceptual side. Parallel processing is a less effective option when you're working

on a laptop, which will be a common setup as a student. Most machine learning teams in industry will have sophisticated parallel processing systems ready for you when you start. It is important to understand how these concepts work, but it's okay to wait until you're on the job to put them into practice. For practice problems, down sampling and batch processing, when available, are usually sufficient.

Back to Table of Contents

Watch: Bias, Variance, and Data Size

At what point is downsampling too much? And do we have enough data as is? Mr. D'Alessandro answers these questions through the lens of the bias-variance tradeoff. The bias-variance tradeoff is a formalized way to understand the competing concepts of model overfitting and underfitting, which are driven by a model's complexity and the amount of data we have to train.

Video Transcript

When we think about the three machine learning failure modes of performance failures, execution bottlenecks, and societal failures, the size of the data set is consistently a common cause of issues. We know that too much data can cause execution bottlenecks and that downsampling is an effective strategy to mitigate that. But at what point did we downsample too much? A similar question is, independent of our need to downsample, do we have enough data as is? I want to answer these questions through the lens of the bias-variance tradeoff. The bias-variance tradeoff is a formalized way to understand the competing concepts of model over- and underfitting, which are driven by a model's complexity and the amount of data we have to train. After we train a model and evaluate its predictions, we are likely to observe some level of error in the predictions. No matter which loss function we use to model error, the error typically comes from two sources. The first source of error is model estimation bias. We could think of model estimation bias as whether or not the given algorithm and feature set we have used to model our problem can capture the true relationship between the features and the label. The other source of error is model estimation variance. We can think of this as, how much does a model's output change if we were to train it on different random samples of our data. These are somewhat abstract concepts. Let me illustrate these with an example made from simulated data. Here is a set of data points generated by a polynomial with one feature and degree of three. The equation that generated the data is shown on the plot, too, so we can see what a polynomial looks like. There's only one input, x, but the output is a combination of x, x squared, and x cubed. Although there was one feature x, the equation uses, again, multiple powers of x, so it results in a non-linear curve. Let's say we want to model this using a linear regression with only one term, which

would be the feature x. In doing this, we're assuming that the true relationship is equal to y equals intercept plus x times a weight. Here is a plot that shows multiple linear curves fit on data from the same polynomial data generating distribution. Each fitted linear regression line is shown on the plot. The only difference between each solid line is the amount of data I use to fit the particular line. We can see that as the sample size goes up, the lines do converge and start to overlap. However, when we compare each line to the dotted line, which represents the true curve, adding more data doesn't change the shape of our fitted curves. In short, a linear model is only an approximation to the polynomial, and no amount of additional data will change this. The difference between the linear model and the third order polynomial is the model estimation bias. Now let's move to an example illustration of model estimation variance. This chart shows six different plots using the same data from the bias example. In the top row, I use a linear model to fit the curves, and in the bottom row, I use a third order polynomial regression. Each column shows an increase in sample size used to fit the models. Within any given plot, I show in gray, the result of fitting a particular model type to many different random samples of data at that same size. The red line in each is the average of the gray lines. There's a lot going on here. But the details of the model estimation variance are revealed with this type of analysis. The dispersion of the gray lines is a good illustration of the variance. The highest variance is in the bottom left. We can generalize this to the following rules. As we get more data, estimation variance goes down. Also, as we use more complex algorithms, estimation variance goes up. Let's summarize this bias-variance discussion and relate it back to our core question, which was, how much data is enough? I have demonstrated that having more data decreases model estimation variance but doesn't improve model estimation bias. However, having more data enables us to use more complex models, such as going from a logistic regression to a decision tree, and that could actually reduce the bias. In short, having more data directly reduces the model estimation variance while it indirectly enables us to reduce model estimation bias. Reducing both the estimation bias and variance is what yields better generalization performance overall. There is no single answer either for how much data is enough. The 'more is better' rule generally applies, but how much more is a question that is problem dependent. You'll always have to answer this question through empirical analysis.

Back to Table of Contents

Watch: Class Imbalance

Now we will look at a particular problem related to performance failures, known as class imbalance. Class imbalance is when one of the classes is rarer in the data. In this video, we examine strategies to mitigate the problem of class imbalance, such as downsampling and upsampling.

Video Transcript

I want to now cover a particular cause of performance failures that is caused by data size, but might be more appropriately characterized as not having enough of the right data. This particular problem is called class imbalance problem. This problem is so common that it is important to be aware of it, understand why it's a problem, and also know how to mitigate it. In words, class imbalance is a situation where one of our classes is much more rare in the data. This usually involves binary cases, and the class that is more rare is usually the positive class, or oftentimes what we label as one. Here is an example illustration of class imbalance. The histogram here shows the distribution of a binary label on two data sets. The balanced label shows a perfect 50/50 split. The unbalanced label is positive only two percent of the time. Technically speaking, we would only consider a data set balance when the label meets an exact 50/50 distribution. But in practice, class imbalance really doesn't become an issue until we're seeing the positive classes less than five percent of the time. Class imbalance is usually created from natural processes and doesn't indicate that there is an error in data collection. One of the most extreme cases would be fraud applications. Think of all the legitimate transactions or website logins that exist. It's pretty common for less than one percent to ever actually be problematic or fraudulent. Additionally, most advertising or commerce applications yield class imbalances. Think of all the ads or promoted products you've ever seen and how infrequently you've actually clicked or purchased one. Despite being quite natural, class imbalances is typically a problem because most supervised learning algorithms need sufficient representation of both classes to learn well. Additionally, when one class is overrepresented, the learning algorithms will implicitly weight the majority class more. When mitigating this problem, we generally want to resample data so that we retain as much of the minority class as possible while reducing the imbalance. We

can approach this, though, both through down- and upsampling. Downsampling is the better choice when you're already starting from a very large data set. Let's say you have 1 million records with a one percent base rate, meaning only 10,000 of those records are labeled positive. If you wanted to take a 20 percent sample to improve computational performance, you can naively just take a random sample. This is good for computational reasons, but this leaves you with only 2,000 positive cases. An alternative approach is to take 100 percent of the positive cases and then take some smaller percent of the negative cases. This is illustrated by the graphic here. This is also referred to a stratified sampling and statistics, which means our sampling rates are conditional on a given attribute of the data. In this case, the attribute is the label. The other strategy you can choose is called upsampling. Upsampling is where we take 100 percent of the negative classes and sample the positive class cases with replacement until we get equal sizes for both. Again here, sampling with replacement means when we sample, we can sample in the same examples more than once. This is depicted by the graphic here. Upsampling is a better strategy when we have limited data to begin with and can't really afford to discard any. These two strategies are simple and effective when your data has class imbalance. However, using these strategies doesn't guarantee prediction performance gains. The approach you use to solve for class imbalances is ultimately a design decision that you can empirically test. If you want to consider these, here are some good rules of thumb to remember. If your base rate is greater than five percent, you probably don't need to make any adjustments. For any base rate below that, it would be smart to consider these strategies. If you want to empirically test different class balancing strategies, always use a validation set that has the original class distribution. Remember that metrics like accuracy, precision, and recall will change if the base rate of your data changes, which ultimately happens when you do upsampling and downsampling. This means, again, that you can't really compare these stats on two data sets with different class ratios. However, when you're testing different class balancing strategies, this is exactly what you're doing. Keep your validation data in its original form and compare different strategies against this data. I'll conclude with an example empirical test on some real data. The chart here shows an empirical evaluation on a binary classification task using logistic regression on an imbalanced data set. This data set has extreme class imbalance, where out of 40,000 examples, there are only 250 positive cases, creating a base rate of about 0.4 percent. Here, I tested upsampling and two downsampling strategies using positive to

negative ratios of 1-1 and 1-10. As a reference, I also compared the three strategies to building a model on the original data. In all cases, my validation set reflects the original label distribution, which again had a base rate of about 0.4 percent. Notice there is a pretty strong improvement when I am using upsampling or a downsampling with a 1-1 ratio. In both of these cases, we achieve perfect class balance. As we introduce more negative cases, it looks like the performance starts to degrade. As I illustrate here, mitigating class imbalance can really improve results, and luckily, this is a technique that we can empirically test to truly understand the benefits.

Back to Table of Contents

Watch: Learning Curve Analysis

In this video, you will be introduced to a method for measuring data size efficiency. This method is called learning curve analysis. Mr. D'Alessandro will walk you through the different elements and then discuss how to build a learning curve. You'll also explore an example based on real data.

Video Transcript

In this video, I want to introduce a straightforward method for empirically measuring data size efficiency. This method is called learning curve analysis. A learning curve is a plot that shows us the relationship between data size and model prediction performance. Here's an example of a hypothetical learning curve. I'll walk us through the different elements and then discuss how to build one. After that, I'll show an example based on real data. The x-axis here represents a particular sample size of data. You can label this as the percent of total records sampled or the absolute sample size. The y-axis is some measure of out of sample model performance. This is a case where the performance improves with more data. The metric here could be AUC or precision. Building a curve like this is pretty straightforward. You would first select a range of sampling rates to cover the x-axis. Usually, you would sample up until your max data size. Then you run a loop where in each iteration you sample from your training data, build a model, and then evaluate this model on your test data. The test data here would have been created prior to the main loop, which you used to build the curve. Here's another example built off of a real data set. I have changed the routine a bit, though. For each sample size that I'm considering, I run a bootstrap process where I fit 50 different models for a given sample size, where each is fit on a different bootstrap data set. In each bootstrap iteration, the data would be different because I sampled from the main training data with replacement. The bootstrap process lets me build multiple models per sample rate, and this enables me to measure the variance of the resulting AUC for each sample rate. You can also see in this case that I'm plotting two curves. For each sample rate, I built a logistic regression and a decision tree. The goal here was to see which method is performing better at a given sample size. This simple curve offers several insights. The first is that decision trees here are generally better for all sample sizes. This was a surprising result to me,

as I usually expect logistic regression to be better with extremely small sample sizes. Notice too, the shape of the curve for each model. The logistic regression reaches its peak performance at around 10 to 20 percent of sampling rate, whereas decision trees keep getting better as we increase the data. This is a good illustration of the bias variance tradeoff in action. Logistic regression is a simple linear model. It has lower variance, which is why it can perform as well with only 10 percent of the data. But no matter how much data we give it, it can't reach the performance levels of the decision tree. This is because of the logistic regression's inherent model estimation bias. When putting learning curves to practice, we can answer two helpful questions. The first is, are you reaching performance plateaus? If your curve looks more like the decision tree than the logistic regression here, then you may want to explore investing in more data. The other question is around downsampling. Smaller data sets are usually always faster to work with. But this comes at prediction performance expense. Learning curve analysis can help you identify the right tradeoff between execution speed and prediction performance.

Back to Table of Contents

Quiz: Check Your Knowledge: Data Sufficiency

You may take this quiz up to three times.

The full contents of this page cannot be rendered in the course transcript. Please complete this activity in the course.

Back to Table of Contents

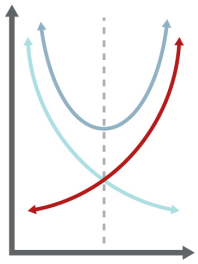
Module Wrap-up: Considerations for Data Sufficiency

On top of selecting good data to work with, there are factors to consider after running your model with your selected data. You can do this by evaluating the results and determine where you can make improvements. Sometimes the model can be improved and sometimes the data can be improved.

In this module, Mr. D'Alessandro explained various techniques to avoid insufficient data and what to do to improve your data. If your model takes too long to execute, you may have a bottleneck issue and need to downsize your sample data. You can also determine whether or not your data is sufficient by building a learning curve. Keep these tactics in mind as you follow and evaluate your future models.

Back to Table of Contents

Module Introduction: Addressing Feature Issues



In this module, Mr. D'Alessandro will further describe another failure mode known as feature issues. Some of the issues may include feature leakage or irrelevant features. Ensuring that you have the right amount and types of features to solve your machine learning problem is crucial because it can easily alter your model results when set improperly. Mr. D'Alessandro will also explain how to resolve these errors using various scenarios as examples.

At the end of this module, you will have the opportunity to practice proper problem formulation where you will first choose and explore your data, build your model, then execute and evaluate your model.

Back to Table of Contents

Watch: Irrelevant Features

Now, we'll be going deeper into failure modes caused by feature issues. It is very likely in all scenarios that at least some of your features will be irrelevant. Mr. D'Alessandro discusses what to do with irrelevant features.

Video Transcript

In this module, we're going to go deeper into failure modes caused by feature issues. Feature problems can certainly cause all failure modes, but in general, we will be more concerned about performance and societal failures. Execution bottlenecks caused by having too many features is easily remedied by feature selection methods. Model performance is heavily dependent on your choice of features. The simplest case to cover is when none of your features have real predictive power. Here's an example way to diagnosis this. The plot here shows future importances from a random forest trained on randomly generated features. By design, there should be no predictive value here. The primary clue here is the test performances. The test AUC is exactly 0.49, which is as good as random. We also see a distinctive pattern in the feature importances, which is they all have the same value. The Random Forest method normalizes the feature importance, so they add up to 100 percent, which is why the importance of each one here isn't zero. But nonetheless, when you see both low out of sample performance and no differentiation amongst the features, you likely need to revisit your feature engineering step and consider adding more. It is hard to say in advance how often you will encounter this exact issue in practice. Most data sets that are created for machine learning practice should have at least a few predictive features, but if you are responsible for engineering features from scratch, then this possibility is more likely. What is very likely in all scenarios, though, is at least some of your features will be irrelevant. Here is a more realistic feature importance plot generated on a real data set. This is built using the cell to cell data predicting churn. The top three features make up about 80 percent of the normalized feature importance, while the last three make up less than five percent. These last three are irrelevant features we can likely remove. Irrelevant features can cause two core problems. First, they increase the risk of overfitting, which then causes prediction performance failures. They also create engineering and maintenance costs. These

costs may not cause what I've been calling execution bottlenecks, but as a general rule, you don't want to incur costs that generate no benefit. So when you do have some irrelevant features, it is best to remove them using standard feature selection techniques.

Back to Table of Contents

Watch: Feature Leakage

In this video, we examine a serious topic: feature leakage. This is a serious problem because it is easy to overlook feature leakage, leading machine learning engineers into a false sense of accomplishment. Let's examine how to detect feature leakage and how to correct it.

Video Transcript

In this video, I'm going to cover a topic that every machine learning engineer must know and that many machine learning engineers in practice may have experienced. This topic is called Feature Leakage and is a particularly insidious cause of model performance failures. The reason I use such a strong word is that feature leakage is very easy to overlook. When it exists, it can lead model developers into a false sense of accomplishment. A more formal definition of feature leakage is when you use information in the model training process that would not be available at prediction time. To really understand this problem, we need to bring ourselves back to the feature engineering phase. When we make predictions, we use historical data to make a prediction at the present moment to predict a future event. When we create a modeling data set, we need to simulate this notion of past, present, and future, but of course, this is all done on historical data. This diagram here illustrates the process. Here's a timeline that is split between historical and future data. When we prepare our data for modeling, the label should be built from events that happen in the future time period. Likewise, all features should be built using data labeled from the historical time period. When your features include data from the time period marked here as future data, you would be committing the error known as feature leakage. Here is a real world example to illustrate the point better. Imagine you're working at an e-commerce company and you want to build a model that predicts whether a visitor to your website is going to make a purchase. The model will be run in real time on the site, and you might use the predictions to customize the offers that you present to the web visitor with each click. Your features will be built from different portions of the website that the customer has already clicked on. Any web click that

happens before a purchase occurs is eligible to use as a feature. Let's assume in your data you accidentally left purchase confirmation pages and use that as a feature. This would be considered leakage because the event happened after we observed the labels, which is the event that someone made a purchase. This is a particularly bad case of leakage because the feature itself would have been caused by the purchase, which is the label. If included in your model, this would lead to exceptional predictive performance in your offline analysis. Remember that when we test for overfitting, we usually see strong training performance but bad out of sample performance. When feature leakage exists, it impacts both our training and our out of sample data set, so it won't be so easy to detect. It may be difficult to distinguish strong performance from feature leakage, from just having a good model. If the strong prediction performance is due to feature leakage, the performance won't be achieved in real applications because you clearly can't use future data in your production models. Detecting feature leakage usually requires a little detective work. The first clue should be really high out of sample model performance. My experience has taught me that when a model's out of sample performance looks too good to be true, it usually is. In particular, if you ever see AUCs greater than, say, 90 percent, you should be suspicious that something weird might be going on. This isn't absolute proof, of course, but it should motivate you to investigate more. The next clue is in the feature importances. Here is an example of feature importance plot built from two models. This is, again, using the cell-to-cell data predicting customer churn. The model on the left has a leaked feature, which is an artificial feature that I created, and the model on the right has the same data but without the leaked feature. The feature on the left labeled leakage is a clear outlier in that chart. When you see a feature with this much relative feature importance, you should take the time to investigate it for potential feature leakage. This is especially true when you see this pattern and also a really high out-of-sample model performance. If feature leakage is suspected, I recommend thinking qualitatively about that particular feature. Ask yourself, does this feature make sense, and would it be available in your production system at the time of a prediction? Also, take a thorough look at your code to make sure you haven't inadvertently made mistakes in handling the timestamps of your data, particularly for future engineering, labeling, as well as splitting your training and test data sets. The feature importances I just showed were produced here using a decision tree classifier. Any tree-based method can give you similar feature importances. I recommend doing feature

importance analysis even if you don't want to use a tree-based method for your final model. As we've seen through several examples I've provided, feature importance analysis can help us identify both irrelevant and leak features, as well as help us create a narrative about how our model is actually working.

Back to Table of Contents

Watch: Concept Drift

Here we look at concept drift, or the changes in the statistical properties of data over time. Mr. D'Alessandro provides some examples and explains how to prevent and remediate concept drift in your models.

Video Transcript

In this video, I'm going to cover a machine learning problem commonly known as concept drift. Concept drift can be formally defined as changes in the statistical properties of data over time. When I say statistical properties, I mean any one of the following: the mean value of a feature or label can be changing over time; the variance of the feature or label can be changing over time; and finally, the expected value of the label, conditional on the features, can be changing over time. Here's an example plot of a feature whose mean and variance is changing over time. This can be any feature, but let's say it is the income of customers that purchase a particular product. This can be increasing because the company introduced more expensive products, which then attracted wealthier customers. The vertical reference line represents the point that we may have collected historical data to build a feature and train a particular model. Over time, both the features' mean and variance changes over time. In general, when our data is being generated by regular, everyday processes, we should always expect that the features will change over time, and the relationships between the features and the label should also change over time. Think about any business over an extended period of time. Most businesses want to grow, and as they grow they take on different types of customers. Think of a service that is initially offered in the U.S. that eventually aims for international expansion. And if that business is successful, other businesses will copy the business model and compete with it. Many forces, such as growth, competition, marketing, and economic cycles will likely change the distribution of the data that you collect. This is just a fact of life, so often can't be avoided. The key is to build systems that are resilient to such changes. This is typically achieved through regular monitoring and retraining of your models. On the monitoring side, in practice, you should build and implement systems that track two things. The first is the mean and variance of your input features, and the second is the actual predictive performance of your model over time. The predictive

performance should be your top concern. If this starts to drop, you'll need to make plans to remediate the problem. The best remediation is usually to just retrain your models on more recent data. When you build a model, whether it is the first implementation or retraining of an existing model, there are strategies you can take to keep the model as relevant as possible to future time periods. Take this hypothetical data that we have represented in this table. Notice how there is a timestamp that we can use to separate the data into five separate periods. These time periods can be months or years, for instance. As a machine learning engineer, we can choose how to create our training data set. When we have a lot of historical data, we need to balance a certain trade off. Using more historical data gives us more data, which we know helps to reduce model estimation variance, which reduces overfitting. However, data that is further away from the current timestamp may be less relevant. This creates a certain type of distribution bias, which is the same bias we encounter with concept drift. We can set up an experiment that will guide us on how much historical data we should actually use. We first set aside the most recent periods for validation and test data. Then each variant of our experiment, we use a different length of historical data, starting from the most recent and then going backwards. This experimental design is depicted in the following diagram. The experiment here shows us how we might train or retrain a model to ensure that the data it uses is the most representational of future data. There is still an open question on when to retrain a model, though. There is no hard and fast rule for this. Retraining a model takes time and development cost, of course. The cost of retraining needs to be balanced against the cost of performance degradation. No matter how you balance these two costs, it is important to set up monitors, so you're at least aware of the presence of concept drift. If you set up your training procedures to be reproducible, then the cost of retraining should be much lower than the original development cost, which gives you better performance over time.

Back to Table of Contents

Quiz: Check Your Knowledge: Feature Issues

You may take this quiz up to three times.

The full contents of this page cannot be rendered in the course transcript. Please complete this activity in the course.

Back to Table of Contents

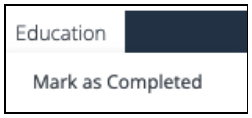
Assignment: Unit 8 Assignment - Choose Your Machine Learning Problem and Data

Over the next two weeks, you will work on solving a machine learning problem of your choosing. You will have three assignments related to this task:

1. In this coding assignment, you will choose one of four data sets to work with. You will also choose your predictive problem and the label. You will use data analysis techniques that you have learned to inspect your data in preparation for developing a project plan in the written assignment that follows. You will begin to think about which model to use and how to prepare your data to build a modeling data set that is suitable for your predictive problem and model.
2. In the written assignment, you will practice proper machine learning problem formulation and draft a project plan. You will describe your predictive problem; your data preparation plan; and your model fitting, evaluation, and improvement plan.
3. In the lab assignment, you will implement your plan and build a machine learning model for your predictive problem. Note that this lab will extend to two weeks.

This assignment will be graded.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. Note: This assignment will be manually graded by your facilitator. You cannot resubmit.

This assignment will be graded by your facilitator.

The full contents of this page cannot be rendered in the course transcript. Please complete this activity in the course.

Back to Table of Contents

Assignment: Unit 8 Assignment - Written Submission

In this part of the assignment, you will practice proper ML problem formulation and draft a project plan. You will answer some questions about the predictive problem you would like to solve and the data set that you have chosen in the coding assignment. You will also explain your data preparation plan as well as your model fitting, evaluation, and improvement plan.

The questions will prepare you for future interviews as they relate to concepts discussed throughout the week. You've practiced these concepts in the coding activities, exercises, and coding portion of the assignment.

Completion of this assignment is a course requirement.

Instructions:

1. Download the Unit 8 Assignment document.
2. Answer the questions.
3. Save your work as one of these file types: .doc or .docx. No other file types will be accepted for submission.
4. Submit your completed Unit 8 Assignment document for review and credit.
5. Click the **Start Assignment** button on this page, attach your completed Unit 8 Assignment document, then click **Submit Assignment** to send it to your facilitator for evaluation and credit.

Back to Table of Contents

Module Wrap-up: Addressing Feature Issues

Sometimes your data may be good to work with, but the features are not. There are various cases that will cause feature issues such as feature leakage and irrelevant features. In this module, Mr. D'Alessandro explained with scenarios how to handle these issues to improve your features and model. These are important factors to consider as you run future models with various data sets with different features.

Back to Table of Contents

Lab 8 Overview

In this lab, you will have two weeks to implement the machine learning project plan that you drafted in the written assignment.

Using a Jupyter Notebook, you will practice exploratory data analysis techniques that you have learned to investigate your data and plan how to prepare your data to build a modeling data set that is suitable for your predictive problem and model. You will then implement your plan and build a machine learning model for your predictive problem.

When you are done implementing your plan, you will create a portfolio by uploading your project and data set to GitHub.

This three-hour lab session will include:

- **10 minutes** - Icebreaker
- **30 minutes** - Week 8 Overview and Q&A
- **20 minutes** - Breakout Groups: Big-Picture Questions
- **10 minutes -** Class Discussion
- **10 minutes - Break**
- **30 minutes** - Breakout Groups: Lab Assignment Working Session 1
- **15 minutes** - Working Session 1 Debrief
- **30 minutes** - Breakout Groups: Lab Assignment Working Session 2
- **15 minutes** - Working Session 2 Debrief
- **10 minutes -** Concluding Remarks and Survey

By the end of Lab 8, you will:

- Load your data set.
- Inspect and analyze the data.
- Prepare your data for your model.
- Fit your model to the training data and evaluate the model's performance.
- Improve the model's performance.
- Create a portfolio to showcase your project.

Back to Table of Contents

Ask the Expert: Mehrnoosh Sameki on Summarizing the Machine Learning Life Cycle

In this lab, you will implement all of the steps of the machine learning life cycle that you learned in this course to solve a predictive problem. You will begin with a business problem and a raw data set, and you will end with a machine learning model that can be used to make predictions about new, unseen data.

In this video, Dr. Sameki summarizes the ML life cycle and your role as an MLE or data scientist in implementing the different steps.

Note: The job title listed below was held by our expert at the time of this interview.

Mehrnoosh Sameki
Product Manager, Microsoft

Dr. Mehrnoosh Sameki is a Senior Technical Program Manager at Microsoft, responsible for leading the product efforts on machine learning interpretability and fairness within the Open Source and Azure Machine Learning platforms. Dr. Sameki also co-founded Fairlearn and Responsible-AI-widgets and has been a contributor to the InterpretML offering.

Question

How to implement the steps in the ML life cycle?

Video Transcript

At a conceptual level, we learn that machine learning is about building a machine that, given a certain set of inputs, will produce a certain desired output by finding patterns in data and learning from it. The data scientist's first job is to collect and prepare data. The quantity and quality of her data dictate how accurate her model is going to be. Once she collects data, she needs to prepare her data. That means wrangling data and preparing it for training, cleaning it, which may require removing duplicates, correcting errors, dealing with missing data, dealing with outliers, data type conversions, and so on. These operations all fall under feature engineering. This is a step a data scientist takes to prepare the proper input data set compatible with the

machine learning algorithm requirements. She might even randomize data, which erases the effect of the particular order in which she collected and/or otherwise prepared her data. She might further visualize data to help detect relevant relationships between variables or class imbalances in case she has not represented a group well in her training data. Now that her data prep step is over, she needs to choose an algorithm for her problem. Of course, there are many different algorithms for different tasks. She needs to choose the right one. For instance, she needs to ask, is this a regression type of problem? Is this a classification problem? Is this clustering or anything else? That helps her identify which class of algorithms she should choose from. If she identifies she's having a regression problem, she now knows she needs to call a linear regression or any other regression algorithm. Same for classification regression, or any other type of machine learning problems. Once she identifies a suitable algorithm, she trains her model on training data. And after the model is generated and trained, she needs to validate the model before using it for making real-time predictions in the real world.

You just heard about the key steps of the machine learning life cycle; here is a summary of those steps for your reference:

1. Collect data, the quantity and quality of which will ultimately determine the accuracy of the model.
2. Prepare and clean that data, which may include removing duplicates and dealing with missing data and/or outliers, data-type conversions, randomization, and visualization.
3. Choose an algorithm, the selection of which is dependent on the type of problem at hand.
4. Train, evaluate, and improve the model before it is used to make predictions in the real world.

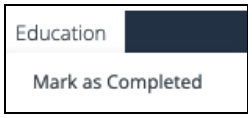
Back to Table of Contents

Assignment: Lab 8 Assignment

In this lab, you will implement your project plan and build a machine learning model for your predictive problem.

This assignment will be graded.

When you finish your work:

1. Save your notebook by selecting the "Save and Checkpoint" entry from the "File" menu at the top of the notebook. If you do not save your notebook, some of your work may be lost.
2. Submit your work by clicking **Education —> Mark as Completed** in the upper left of the Activity window 

3. Note: This assignment will be manually graded by your facilitator. You cannot resubmit.

This lab assignment will be graded by your facilitator.

The full contents of this page cannot be rendered in the course transcript. Please complete this activity in the course.