Machine Learning Foundations Unit 5: Evaluate and Deploy Your Model

Table of Contents

- Watch: Heuristic Feature Selection
- Assignment: Performing Feature Selection Using Scikit-Learn
- Watch: Stepwise Feature Selection
- Watch: Using Regularization for Feature Selection
- Module Wrap-up: Feature Selection

Module Introduction: Choose Model Evaluation Metrics

- Watch: Confusion Matrix and Classification Metrics
- Read: Classification Evaluation Metrics
- Code: Confusion Matrix Demo
- Read: Choosing a Classification Threshold
- Watch: Evaluating a Classifier Using the AUC-ROC Curve
- Read: AUC-ROC Curve
- Ask the Expert: Ask the Expert: Kathy Xu on Probabilistic Models
- Module Wrap-up: Choose Model Evaluation Metrics

Module Introduction: Deploy Your Model

- Code: Making Your Model Persistent
- Ask the Expert: Ask the Expert: Kathy Xu on Using ML Models in Production
- Read: Deploying, Hosting, and Monitoring Your Model
- Tool: Deploying, Hosting, and Monitoring Your Model
- Ask the Expert: Ask the Expert: Francesca Lazzeri on Deploying to the Cloud
- Ask the Expert: Ask the Expert: Miriam Vogel on Mitigating Algorithmic Harms After Model Development
- Tool: Upload Jupyter Notebooks to GitHub Repository
- Assignment: Unit 5 Assignment - Model Selection for KNN
- Assignment: Unit 5 Assignment - Written Submission
- Module Wrap-up: Deploy Your Model
- Live Labs: Lab 5 Overview: ML Life Cycle: Evaluation and Deployment
- Assignment: Lab 5 Assignment

Homepage

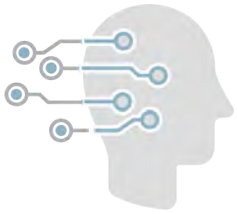
Video Transcript

If you're here watching this video, you're probably already aware that artificial intelligence and machine learning are important technologies. I'd go as far as saying they are critical in today's digital society. Nearly every action in today's economy involves data, and AI helps organizations more objectively and efficiently perform the core tasks, such as delivering content or assessing risk. AI can and will impact us all as consumers in this modern economy. All of the AI is built by a small set of people with highly specialized and sought-after skills. Given the breadth of AI's reach, it is important that the people building it are able to understand the diverse perspectives and lived experiences of the people it affects. As the use of AI matures, we're hearing more and more about how AI might be used as a tool for oppression, silently capturing and reinforcing society's biases. But AI-driven problems can be contained and constrained with responsible and inclusive design and testing guardrails that look for potential problems. This is an important part you will play and why diversity in the people who build AI systems is critical. Diversity in tech is how AI gets built for the people by the people.

What you'll do

- **Understand the machine learning life cycle and explore common machine learning packages**
- **Perform exploratory analysis to understand your data and prepare your data for machine learning applications**
- **Train and optimize two popular supervised learning algorithms: k-nearest neighbors and decision trees**
- **Understand the mechanics of linear models and implement a common linear model from scratch**
- **Define the model evaluation metrics for specific applications by selecting the appropriate model candidates and hyperparameters for testing**
- **Understand the principle of ensemble models and how to use them to improve model performance**
- **Explore the fields of computer vision and natural language processing then implement deep learning models to solve problems in these areas**
- **Identify performance issues and societal failures then find solutions to address these issues**

Course Description



Machine learning (ML) is increasingly a part of people's daily lives. Think about some of the technologies you use every day, such as the suggestions that appear on YouTube and emails being diverted to the spam folder. All are practical applications of ML, a branch of artificial intelligence (AI) that allows computer programs to automatically improve through experience. Looking ahead, some of the world's most complex problems — such as future pandemics — could depend on ML as a solution.

With a curriculum taught by Cornell Tech's Visiting Lecturer Brian D'Alessandro, with content from Cornell University and Boston University faculty, and developed in concert with industry leaders, this "Machine Learning Foundations" summer course offers you the skills that will enable you to build ML solutions in real-world conditions through an ethical and inclusive lens. In this skills-based program, you will work with industry-relevant tools to analyze real-world data sets to identify patterns and relationships in data. By the end of this nine-week course, you will have hands-on experience solving real-world problems by working through the ML life cycle to build machine learning models.

Weekly synchronous lab sessions will give you the opportunity to explore these skills in a collaborative environment and gain hands-on experience in machine learning and data science. Ultimately, the foundational skills you acquire in this "Machine Learning Foundations" curriculum will prepare you to take on real-world industry challenges in Studio this fall.

During this part of the **Break Through Tech** machine learning program, you can expect to spend about eight to ten hours per week on asynchronous online content and about three hours per week on synchronous lab sessions with other students. For a detailed list of what is required to successfully complete this course, visit the "Syllabus" link under **Course Resources** in the left navigation menu.

Brian D'Alessandro
Head of Data Science, Social Impact
Instagram

Brian D'Alessandro is a practicing data science executive with 20 years of experience building machine learning and statistical models for industrial decision making. Mr. D'Alessandro currently leads data science for Instagram's Well Being and Integrity teams. Prior to Instagram, he held leadership roles at Capital One, Zocdoc, and Dstillery. Within these roles, Mr. D'Alessandro led the development and execution of AI and

ML systems enhancing digital experiences serving healthcare and financial service needs. Mr. D'Alessandro has also served as an adjunct professor for NYU's Center for Data Science Master's of Data Science program. He developed and taught core ML classes for incoming Master's students and has helped over 1,000 students start their careers as data scientists and machine learning engineers.

Dr. Kilian Weinberger
Associate Professor
Computing and Information Science, Cornell University

Dr. Kilian Weinberger is an Associate Professor in the Department of Computer Science at Cornell University. He received his Ph.D. from the University of Pennsylvania in Machine Learning under the supervision of Lawrence Saul and his undergraduate degree in Mathematics and Computer Science from the University of Oxford. During his career, Dr. Weinberger has won several best paper awards at ICML (2004), CVPR (2004, 2017), AISTATS (2005), and KDD (2014, runner-up award). In 2011, he was awarded the Outstanding AAAI Senior Program Chair Award, and in 2012 he received an NSF CAREER award. Dr. Weinberger was elected co-program chair for ICML 2016 and for AAAI 2018. In 2016, he was the recipient of the Daniel M. Lazar '29 Excellence in Teaching Award. Dr. Weinberger's research focuses on machine learning and its applications; in particular, he focuses on learning under resource constraints, metric learning, machine-learned web-search ranking, computer vision, and deep learning. Before joining Cornell University, Dr. Weinberger was an Associate Professor at Washington University in St. Louis and had previously worked as a research scientist at Yahoo! Research.

Dr. Linda K. Nozick
Professor and Director of Civil and Environmental Engineering
College of Engineering, Cornell University

Dr. Linda K. Nozick is Professor and Director of Civil and Environmental Engineering at Cornell University. She is a past Director of the College Program in Systems Engineering, a program she co-founded. Dr. Nozick has been the recipient of several awards, including a CAREER award from the National Science Foundation and a Presidential Early Career Award for Scientists and Engineers from President Clinton for "the development of innovative solutions to problems associated with the transportation of

hazardous waste." She has authored over 60 peer-reviewed publications, many focused on transportation, the movement of hazardous materials, and the modeling of critical infrastructure systems. Dr. Nozick has been an associate editor for Naval Research Logistics and a member of the editorial board of Transportation Research Part A. She has served on two National Academy Committees to advise the U.S. Department of Energy on renewal of their infrastructure. During the 1998-1999 academic year, she was a Visiting Associate Professor in the Operations Research Department at the Naval Postgraduate School in Monterey, California. Dr. Nozick holds a B.S. in Systems Analysis and Engineering from George Washington University and an M.S.E and Ph.D. in Systems Engineering from the University of Pennsylvania.

Dr. Chris Myers
Senior Research Associate and Adjunct Professor
Center for Advanced Computing, Cornell University

Dr. Chris Myers has been with the Center for Advanced Computing at Cornell University since 2017, having previously been a member of the research staff of the Bioinformatics Facility of the Institute of Biotechnology (2007-2017) and the Cornell Theory Center (1993-1997, 1998-2007). In addition, he is an Adjunct Professor in the Department of Physics at Cornell and a member of the graduate faculty in the fields of Physics, Computational Biology, Applied Mathematics, and Computational Science and Engineering. Dr. Myers works primarily in the field of computational biology, addressing problems in the systems biology of cellular regulation, signaling, metabolism, development, virulence and immunity, and in host-pathogen interactions and the spread of infectious diseases on populations, networks, and landscapes.

Dr. Mehrnoosh Sameki
Adjunct Professor, Boston University
Senior Technical Program 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 platform. She co-founded Fairlearn and Responsible-AI-widgets and has been a contributor to the InterpretML offering. Dr. Sameki earned her Ph.D. degree in Computer Science at Boston University, where she

currently serves as an Adjunct Assistant Professor offering courses in responsible AI. Previously, she was a data scientist in the retail space, incorporating data science and machine learning to enhance customers' personalized shopping experiences.

Unit 5: Evaluate and Deploy Your Model Overview Video Transcript

At this point in the course, we've learned how to formulate a machine learning problem, how to prepare data for machine learning, and how to execute your first model. We are close to completing a full loop of the model development lifecycle. Our next enterprise will be to learn how to tell if your data and models are performing to your expectations. We will go deeper into various lost functions and metrics that are traditionally used for model evaluation. We will also cover how to achieve our core goal of good model generalization. In this series, we will learn how to prepare and evaluate data with good generalization in mind with a focus on getting optimal performance while avoiding overfitting. Our aim is for you to be able to go beyond just training a model and instead be able to demonstrate with confidence that your selected model can effectively solve the problem at hand.

What you'll do:

- **Understand the importance of model selection in machine learning**
- **Choose model evaluation metrics that are appropriate for the application**
- **Choose appropriate model candidates and hyperparameters for testing**
- **Set up training/validation/test splits for model selection**
- **Apply feature selection techniques to get a better-performing model**
- **Explore how to deploy, host, and monitor your model**

Unit Description

In this unit, you will be introduced to the fifth and sixth steps in the machine learning life cycle: evaluation and deployment.

Now that you have trained your model, how do you know whether it will generalize well to new data? You will be introduced to techniques that can be used to properly evaluate and improve your model's performance with a view toward producing the best model for your data and ML problem.

You will explore different model selection methods that are used to find the best-performing model. Mr. D'Alessandro will discuss common out-of-sample validation methods that are used to test a model on unseen data in support of model selection. You will discover how hyperparameter configurations as well as the combination of features play a role in model

performance. Using your own implementation along with built-in scikit-learn libraries, you will determine the optimal hyperparameter configuration for your model and perform feature selection techniques to identify the combination of features that results in the best model performance.

Once you have trained and tested your model, it is time to make it available to stakeholders to solve the business problem for which the model was developed. You will explore different ways of deploying your model as well as things to monitor once your model is in production.

Note: The code snippets in videos may vary from the code in assignments and exercises.

Back to Table of Contents

Tool: Unit 5 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 5 Glossary** as a reference as you work through the unit and encounter unfamiliar terms.

Back to Table of Contents

Discussion: Unit 5: Evaluate and Deploy Your Model

Welcome to the discussion thread for Unit 5! This is a space where you can ask any questions you have about evaluating and deploying models

To get the most helpful responses from your peers and instructional teams, be specific about the concepts you're struggling with and share what you've understood so far.

Guidelines for Posting in Discussion Threads:

1. **Stay On Topic:** Make sure your questions and contributions are relevant to the unit's content. This helps keep discussions focused and beneficial for everyone.
2. **Be Specific:** When you post a question, include specific details and context. Mention particular concepts, lectures, assignments, you're referring to. This makes it easier for others to provide accurate and helpful responses.
3. **Quote or Cite Materials:** If your question is about a specific part of the course materials, quote or cite the section to help others understand and reference the context of your query. You can even use a link to the specific course page!
4. **Constructive Language:** Use respectful and constructive language in all your interactions. Encourage and appreciate responses, and offer positive feedback when others help you.
5. **Use Clear and Concise Language:** Keep your questions and comments clear and to the point. Avoid overly complex language that might confuse your peers or instructional teams.
6. **Collaborative Learning:** Encourage discussion by not only asking questions but also responding to others. Sharing your insights and understanding can contribute significantly to collective learning.
7. **Confidentiality and Respect:** Do not share personal information or content that hasn't been publicly discussed in the course materials. Always respect the privacy and intellectual rights of your peers and instructors.
8. **Search Before Posting:** Before posting a question, check if someone else has asked something similar and whether it has already been answered. This helps reduce duplicate

questions and leverages existing discussions.

9. **Use Proper Formatting:** If you're sharing code, equations, or specific formatting-dependent content, use the appropriate formatting tools provided by the discussion platform to enhance readability.
10. **Review and Edit Before Posting:** Before you submit your post, review it for clarity and errors. A well-written post is more likely to receive useful responses.
11. **Maintain Academic Integrity:** Do not post exact answers to assignment questions. Instead, focus on discussing concepts and problem-solving approaches. This helps preserve the integrity of the learning process and ensures everyone benefits fairly from the course materials.

Back to Table of Contents

Module Introduction: Introduction to Model Selection

Now that you understand how to train a machine learning model, the next step to explore is evaluating and improving your model to ensure it is the right one for your problem. In this module, Mr. D'Alessandro will describe the approach to selecting an optimal ML for a problem. You will be introduced to the concept of out-of-sample validation, which is an important technique used for evaluating a model's ability to generalize to new data. Mr. D'Alessandro will also explain best practices for searching through a set of candidate models with varying complexities in order to narrow in on the best-performing model.

Back to Table of Contents

Watch: Out-of-Sample Validation Overview

You previously used scikit-learn's `train_test_split` function to split a data set into training and test data sets. Evaluating your model on test data is known as out-of-sample validation. In this video, Mr. D'Alessandro explains the motivation behind splitting your data set. You will discover how out-of-sample validation is a key step in developing viable machine learning models that achieve good generalization. To help you solidify your understanding, Mr. D'Alessandro further demonstrates the idea of out-of-sample validation with a practical scenario using Netflix as an example.

Video Transcript

In this video, we're going to set up the motivation for the specific processes useful for use for model evaluation and supervised learning. We'll see that there are many evaluation metrics to choose from. But no matter the metric we use, there is one fundamental principle will always need to stick to. This principle is called out-of-sample validation which simply just means that when we evaluate our models, we must use data that was not used in model training. This does not mean that you can't evaluate your training data. Often, it is helpful to do so for debugging purposes. But when our ultimate modeling goal is generalization, we can only really trust the metrics computed on non-training data. The reason we care so much about out-of -sample validation is that we cannot fully trust our training data. To make this point, I need to get a little conceptual and theoretical. Let's imagine there are some oracle that creates our data. We'll call this the data generating process. Mathematically, we could represent this as some probability distribution over a set of features. We represent this theoretical data universe here in the left hand chart. Now in reality, we don't have access to this theoretical data set. Instead, we can only observe some sample of that data. This sample would be represented by the right hand side of this chart. Now in this example, the red dots on the right here were sampled from the black dots on the left, so in theory, they're representation of the same data generating distribution. But due to randomness in the sampling process, we get a slightly different distribution empirically from the theoretical data set. The error or loss that we want to optimize is on the theoretical data set, and we'll call this the expected loss. Due to the inherent randomness in the data, the loss we often observe in training will be different and often a little too optimistic than from the expected loss. This discrepancy between expected loss and training loss is one of the biggest challenges we'll face when developing models. Most of our validation procedures are designed to help us mitigate this risk. Let's use an example to make this a little more realistic. Let's say you're a

machine learning engineer in Netflix. The business model depends on subscriptions so one important modeling task would be to predict who is likely to keep their membership for at least a year. You might want to use this for marketing purposes so that you can find future profitable customers. The theoretical data universe here would be all of the potential customers in the world with a special emphasis on future customers. But we can't observe these future customers at the time of model development so our observed data that we can use for training only includes past customers which is a subset of the full data universe. Most models are applied on future events. This example is pretty representative of most use cases for machine learning. Another way of thinking about generalization is we want our models to perform well on future events. We should care that our model performs well on past events but only if that performance translates to future event performance. The error or loss we want to optimize again on the theoretical data is called the expected loss. The way we solve this problem is to create a simulation of the full data universe by splitting the data we do observe. This diagram shows a simple but sufficient method for doing this. The main point here is we need to always set aside some of our observed data into what we call a test set. This test data is our approximation of the full data universe. We cannot compute expected loss for the reasons already explained: the data is technically theoretical. But we can approximate it and that is what the test data is for. It is highly likely that the training loss will be different from the test loss. This will happen either through over or underfitting on the training data. The process of model selection aims to search this base of model design options to find the perfect balance between over and underfitting. When we hit that spot, we'll end up with a model that hopefully optimizes the expected loss, which again means it will generalize well on future data.

Back to Table of Contents

Watch: Model Selection Overview

Developing a viable machine learning model that generalizes well requires a search through a set of candidate models. In this video, Mr. D'Alessandro explains the approach for determining the best-performing model from a set of candidate models with varying complexities. In addition, Mr. D'Alessandro discusses how model complexity depends on three specific factors as well as the tradeoff between training time and test set performance.

Note: In the plot at 5:16, the line at the top is red, the middle line is green, and the bottom line is yellow.

Video Transcript

Our true goal when building a supervised learning model is to optimize our out-of-sample loss. We do this through a process that we generally call model selection. Selecting the best performing model is strictly an empirical process. This means we're not using any formulas or theory to design the best model in advance. Instead, we choose a set of candidate models, loop through them all, and select the one that best meets our performance criteria, which is usually some optimal measure of loss on our test set. Successful model selection is driven by two key criteria. The first is that you need to be able to identify a good set of candidates to try out. We can think of model selection as a type of search algorithm, and if your search isn't varied enough, you're likely to miss out on some really good candidates. The second criteria is that you need to apply a good out-of-sample evaluation technique. Of these two criteria, the first is definitely more subjective. This is where your knowledge and intuition becomes your guide, so it will be important to understand the high level concepts that inform this decision. The second criteria is absolutely important to get right, but there are only a few straightforward methods to use. These methods are even implemented in scikit-learn, making it easy to consistently get it right. When it comes to designing the appropriate candidate set, the most important thing to consider is the diversity of options when it comes to model complexity. This chart here shows a hypothetical but realistic performance curve. The x-axis represents model complexity and the y-axis is your test loss. What we typically find and should expect is that, there is a middle ground where the model is flexible enough to learn subtle patterns in the data but isn't so flexible that it overfits. When you design your candidate set, you need to be able to identify candidates that span the full range here, represented by the x-axis. The challenge is that we usually can't reduce the model down to a single complexity score. This means you'll have to use subjective judgment. We can however produce plots like this one on our actual data. When we see this characteristic shape, we can

feel more confident that we've chosen a sufficiently broad set of model candidates. If our plot shows only one direction of movement, which I have labeled here as too-narrow exploration, you would want to revise and expand your candidate set. Now I want to get specific on what it means to design a set of model candidates. The three dimensions we need to consider are the algorithm, the features, and the hyperparameters. Each model candidate will be specified by a single algorithm type, a single subset of features, and a set of hyperparameters that are specific to the given algorithm. Here is an example of a model candidate set represented as code. The top block shows four examples of an algorithm and a specific set of hyperparameters chosen for that algorithm. Notice that we're only using two algorithms, but within each using two different hyperparameter configurations. The second block shows two candidate feature sets, here we're using either all features or half of them. This is just an abstract example so the features are labeled X1 to X6 and they don't really have any meaning here. But in the last block, we loop through the model candidates and the feature sets. We'll call the fit method to build a model predicting our theoretical label Y. Each iteration gives our full model specification along three design dimensions. I want to revisit the model complexity versus test loss curve, but now I'll present how our candidate selection fits into the curve. Remember, we generally can't reduce a model candidate down to actual complexity score. There are some general rules to consider though. As is shown in the red text boxes, each design dimension will enable us to span the low versus high complexity range. When it comes to the algorithm, the simple rule is that linear models tend to be low complexity, and even though KNN or decision trees are considered relatively simple models themselves, they're actually very flexible and can have high complexity. The number of features also relates to model complexity. More features gives the algorithm more opportunities to fit the data, which again is how we define model complexity. Last and likely the most important consideration is the hyperparameters we choose. The hyperparameters restrict or enable the flexibility of the given algorithm. Even if your candidates only use one algorithm, like decision trees, you can span the full complexity range, just through the hyperparameters. This is something you'll need to always be mindful of. Now let's cover one last point. When we design our candidate set, we need to determine how many candidates is enough. This is another case where we can't say in advance exactly what the right number is. We instead need to use our subjective judgment and we need to find a middle ground between two extremes. The horizontal lines here represent different sets of model candidates. The specific designs don't matter to make the point. In the red line we only evaluate three different candidates. We can see here that our range is not sufficient enough for us to get close to the optimal point. The yellow line doesn't have this problem. We have a lot of candidates so we probably do best at finding the optimal point. But this comes at a cost. Each point here requires training a model and depending on the data size, this can be

an expensive process. The green line here is the compromise, it has enough candidates to cover the full range of model complexity but has about half the training cost. The choice of how many candidate models to test ultimately becomes a trade off between training time and test set performance. The exact balance of this trade off will depend on elements of your problems such as the data size, your computing environment, and how much time you have to actually finish the project.

Back to Table of Contents

Module Wrap-up: Introduction to Model Selection

In this module, you discovered the key steps for selecting an optimal model for a machine learning problem. Mr. D'Alessandro introduced out-of-sample validation and explained how it is used to assess the ability of a model to generalize. He also described how model complexity ultimately depends on three different factors: the algorithm itself, features, and hyperparameters. Keep these factors in mind when you are solving future machine learning problems with the goal of finding the model with an ideal complexity.

Back to Table of Contents

Module Introduction: Perform Model Selection

Each machine learning algorithm has its strengths and weaknesses. In this module, Mr. D'Alessandro will show you how to choose the best model for a data set and how to implement out-of-sample validation in practice. He will also explain how various sampling strategies are used to support the model selection process. This basic knowledge will prepare you to complete two practical coding exercises in which you will practice cross-validation and model selection.

Back to Table of Contents

Watch: Using LLMs (Part 1) - How To Use?

In this video, Mr. Yurochkin shows how you can use in-context learning with LLMs. He demonstrates how to give an LLM an example of an input and output that you want. We can also use LLMs to generate synthetic data and use sampling to control the creativity of the responses.

Video Transcript

So how do we use this model? So as I mentioned, the Bayes model. So the Bayes LLM is not-- it's not as nice and easy to use as ChatGPT where we can just simply ask it something and get an answer. But it's still actually very, very useful.

And so, of course, the direct use is just an x token prediction. So let's say if we type the capital of France is, it will now say Paris. But again, not as exciting. So the way to get these models to work is in context learning.

So let's see some examples. So suppose I want to ask a language model. So here, Falcon $40 billion model. So this is one of those pre-trained language models that is open-source. So we're asking it a question about k-12 public schools. And so we give it these multiple answer choices. And we are hoping that the model will actually choose one of the answers. Basically answer the question as a human would.

By the way, so this example here is based on the IBM's server for using different large language models. But that's not what happens here. So the model just continues. It keeps going. So it was trained to predict the next token, and that's what it attempts to do here, pretty much.

So what if we change the prompt a bit? So one thing that's usually good to do is to add a bit of structure. So we can add those tags user, and then put what we want to ask it, and then say answer. And hopefully, it kind of gets a clue that it needs to answer.

Well, unfortunately, it still doesn't happen here. So this is generally a good thing to do, but it's not enough by itself. So the secret sauce to getting this Bayes large language models to work is in-context learning. So it means that we provide an example of what we want it to do as part of our input.

So in this case, we give it an example of a question. So, OK, what is the capital of France? We give it some options-- London, Hamburg, Paris, and then we say answer Paris. So now we set

up the task. So we want to do question answering. We also set up the structure.

So we have this. The prompt is very structured so that the user answer in a particular way of answering the question. So now we can go back to asking the question we originally were interested in about the k-12 public schools. And so now we can see that the model actually does answer in exactly the same format as was prescribed.

So this way, we can get these models to do all kinds of things as long as we can just give it an example of an input and the output that we want. And now we can start using it for all the similar tasks, which follow the same format.

There is still one little problem here. So after giving us the answer, the model kind of keeps going and start generating some stuff that we don't really need and it doesn't make sense. So this is very easy to address. So there is this notion of a stopping criteria.

So if you prescribe the structure through a prompt, so you know that in our case, once the model generates a new line, we no longer-- we know that it has finished its answer. The answer is there. So we can just tell the model to stop after a new line, after the first new line.

So the other very exciting use case of these Bayes large language models is generating synthetic data. So we talked about how traditional machine learning mostly suffers from this need to collect data, label it. So now we have such these large fancy models, and we can use them to make a lot of data ourselves just using the model. So without much human effort.

So in this example, we ask the model to write 10 questions about education. And we again need to provide some examples to it. So we write the first three question ourselves. And then we let the model to generate the remaining seven.

So if you look at the questions here, they're a little bit boring. So they all start with like, what are some of the most something? Effective strategies for pressing issues, promising solutions, effective ways. So while those are valid questions and they could be useful if there is not a lot of diversity here.

So this brings us to another important aspect of the Bayes large language models, but also just large language models in general, which is how they create the text. So when a language model creates a text, it does so word by word. So it looks at what it has seen so far, including what it has generated, and it makes a decision what should be the next word.

So it does so by basically assigning the probabilities to all possible words that it knows. So let's say when you say the capital of France is, it might say Paris with 98, and then maybe London with 2% because this is how it operates, or sometimes in a more ambiguous

situation. So you can say I like, and then the language model can think that the next word would be, let's say, hamburgers with some probability, French fries with another probability, and so on.

And so it needs to-- so how should it decide which word to actually present to a user? Well, so the obvious strategy is to use greedy decoding. So this is the one that we used to generate those 10 questions before. So greedy means that it always picks the word with the highest probability.

But while sometimes it's good when it comes to synthetic data generation, we want to make it a little bit more creative. R Otherwise, we end up with those very like similar kinds of questions. And so the way to do it is to enable sampling.

So instead of now, let's say, asking the model to just always output the word with the highest probability, we actually use the probability distribution to-- it's kind of like a flipping a coin or throwing a dice, where each side has a different likelihood. So the dice is unfair, unbalanced.

So now instead of actually just always looking at the unfair side, we actually throw the dice. And even though very often it will come on the unfair side, it will sometimes also come on different sides. So this is achieved during sampling, through sampling.

So here we see that the questions we start getting, they are a lot more diverse, I should say. So let's say they don't always start with what are some of the most something. Now we have questions such as like, should schools have a dress code? So the questions look more diverse.

And we can actually-- so there is also a parameter called temperature. So temperature essentially says, OK, how creative we want our language model to go. So when we push our temperature to a high number, it basically we try to equalize all the probabilities.

So in the extreme case of temperature being very large, the language model stops being a language model, and every word becomes-- starts having the same probability. If we push temperature to zero, we actually go back to the greedy decoding. So zero temperature means greedy. And very large temperature means very, very diverse.

But here let's say the example here, we set the temperature to 1.5 and it is a little bit too creative, I would say. So it generates something about-- the questions about education, as we intended, but then it starts talking about COVID.

And then suddenly at the end of the day, it generates something about online dating. So which is not quite relevant to the context. So this is something to play with. So you can think

of temperature as this creativity parameter, and which is especially useful when you try to create this synthetic data set.

Back to Table of Contents

Watch: Creating Training, Validation, and Test Sets

Recall that one of the key principles for out-of-sample validation is to ensure our training set and test set are independent of each other and that the test set is representative of future data. These two principles can be achieved with the proper sampling strategy. Here, Mr. D'Alessandro goes over three sampling strategies that can be used to ensure both principles are met.

Video Transcript

When we do out-of-sample validation, we need to make sure that the data we use is completely independent from the training data, and it is representative of the data that the model will be applied to. Both rules help ensure that our out-of-sample data will not lead us to an overfit model, and the model will generalize to new, previously unseen data. We'll satisfy both these rules by thoughtfully sampling our data. Here, we'll cover three different sampling strategies to consider. First, let's discuss the scenarios that dictate which strategy to use. There are two key questions you need to ask about your data. These are, do the units of your analysis appear only once in the data or do they appear in multiple rows? The second question asks is there a time dimension? The table on the top here shows example data where each user ID appears only once and they all have the same timestamp. This is the simplest scenario, and answers "no" to both questions. The bottom table shows example data where each user ID has a record for multiple timestamps. This table answers "yes" to both questions. When the answer to both of these questions is no, we can just pull a simple, random selection of all examples into a test set. This code snippet shows how we might do this on a data frame. First, we specify how much data we want to split into a test set. Then we generate a random sequence to use as our filter. The last two steps use the common data frame filtering logic to create a train and a test set. The next scenario to consider is when we have multiple examples of the same unit ID. In this case, we cannot just simply randomly sample the examples into a training and a test set. If we did this, we would have examples from the same unit in both the training and out-of-sample data sets. This would violate the independence rule we had and would lead to overly optimistic out-of-sample validation. The right strategy here is then to randomize at the unit level. Here is one example implementation of this strategy. In this example, we use a combination of hashing and modulo functions to map each record into a sampling bucket. The bucketing function here is deterministic, meaning that if we put the same input in, we'll always get the same output. If we apply this to each example, we'll put all of the instances of the same unit ID into the same

bucket. We can then randomly sample the records based on the hash bucket. This guarantees that examples with the same unit ID will only be in one of the training validation or test data sets. When the data has a time dimension, we want the latest data to be our test and validation data. This is because we want our model selection and out-of-sample loss estimation to represent future time periods and usually the latest data we have serve as the best proxy for future data. The logic for doing this is straightforward. You would first choose a specific date and everything before that date can be your training data. The rest will be for validation and test. When we answer yes to both of the questions, we would want to apply both strategies. In this case, we would first randomize the units as we did before, with each unit assigned to separate buckets. We would then take the latest examples of those units assigned to the out-of-sample test data sets. With this design, our models would be the most ready to take on new, previously unseen data.

Back to Table of Contents

Quiz: Check Your Knowledge: Split Your Data Set

You will be given three machine learning application scenarios. Determine how you would split the data effectively into training and validation sets for the purpose of training and evaluating a model.

You may take this quiz up to three times.

Please complete this activity in the course.

Back to Table of Contents

Read: Splitting Data Sets for Model Selection

Out-of-sample validation refers to using data that a model was not trained on to evaluate the model. Out-of-sample validation is performed so as to properly evaluate how well a model generalizes to new, previously unseen data.

When performing model selection to choose an optimal model for our data and predictive modeling problem, we will usually split the initial data set $\boldsymbol{D}$ Into three mutually exclusive subsets:

- $\boldsymbol{D}_{\mathbf{TR}}$ — training data set
- $\boldsymbol{D}_{\mathbf{VA}}$ — validation data set
- $\boldsymbol{D}_{\mathbf{TE}}$ — test data set

A common choice is to split the data 70/20/10, respectively. You use the training data set for model fitting, the validation data set to perform model selection, and the test data set for the final evaluation of the selected model's performance.

☆ Key Points

When performing model selection, you should split your data set into training, validation, and test sets to properly evaluate model performance and prevent overfitting

A common choice is to split the data into a 70/20/10 split

There are different methods used to determine how the data should be split into the three subsets

How do you split a data set?

You have to be careful when splitting the training data into the three sets. All three sets should be drawn from the same data distribution. A good rule of thumb is to make sure that the data sets simulate the real-life settings in which the algorithm will operate. There are different methods that can be used to split your data into different sets; the most common is uniformly at random. If the data has a temporal component and changes over time, however, you should split the data by time to make sure that you never predict the past from the future and always the future from the past.

An example of such a data set with a temporal component is email spam. Spam emails change over time, as spammers adapt their emails to get past spam filters. What makes spam filtering so challenging is that you don't know what clever tricks the spammers will

deploy tomorrow. To address this problem, you could collect data for a few weeks to train your model. Next, collect data for a few days to use as your $\boldsymbol{D_{\mathrm{VA}}}$ data set to validate your model. Finally, collect data for a few more days to generate your $\boldsymbol{D_{\mathrm{TE}}}$ data set to test your model. This process is essentially simulating the real-world setting that a spam filter trained today should catch the spam emails of tomorrow.

Why is validation data important?

A common scenario is that the first machine learning algorithm that is trained on $\boldsymbol{D_{\mathrm{TR}}}$ does not perform well enough on $\boldsymbol{D_{\mathrm{TE}}}$ and needs further refinement. For example, imagine you are building an email spam filter with the objective to catch at least 99% of all spam, with at most 1/1000 false positives. If the initial algorithm is not accurate enough for your needs, you may have to use a different algorithm or change its hyperparameters. The problem is that you cannot tweak your algorithm to perform well on $\boldsymbol{D_{\mathrm{TE}}}$; otherwise, you would be overfitting your model to this specific data set and not necessarily improving its accuracy on new data. The error obtained on $\boldsymbol{D_{\mathrm{TE}}}$ will only be an unbiased estimate of the true generalization error of the model if the model has been trained independently of this test set. The moment you look at the test data once and make changes to the algorithm, it is no longer independent. This is where the validation data set comes in.

The validation data set is a proxy for the test set. In practice, you train your algorithm on the training set and evaluate it on the validation set. If your model is not satisfactorily accurate, you continue tweaking it until the validation error improves to an acceptable level. You then complete a single and final evaluation of your model on the test set $\boldsymbol{D_{\mathrm{TE}}}$ to find the unbiased estimate of the generalization error of your final model. Often, the final model is re-trained on the union of $\boldsymbol{D_{\mathrm{TR}}}$ and $\boldsymbol{D_{\mathrm{VA}}}$ so as to not waste the examples in the validation set.

Back to Table of Contents

Watch: Cross-Validation

In cases where the data size is too small, the choice of sampling technique becomes even more important. Follow along as Mr. D'Alessandro explains the drawbacks of having a small data size, along with the method you can use to combat this problem. He also introduces an approach called k-fold cross-validation, which is a creative data-splitting scheme that allows you to reuse training examples when the training set is small.

Video Transcript

Now I'm going to cover some really practical and important decisions, which is what should the relative sizes of the training, validation, and test data sets be? Then I'll follow with, what do you do when these standards splitting procedures don't give you enough data for training, validation, and test? Let's assume we need a validation set and a test set since we're usually going to have to do model selection. The standard rule of thumb is to split your data by a 70, 20, and 10% ratio. Where 70 is for training, 20 is validation, and 10 is for test. This is not a hard and fast rule, so you have to apply judgment here. In all three data sets, the sample size affects the variance of what you're estimating. This means that more data leads to less error. You want to have sufficient data in each bucket, so that means compromising one bucket to benefit the others. Training a model usually requires the most data, so you want the majority share to go there. When you have a lot of data to work with, say, tens of millions or more examples, which is fairly common in a lot of companies, these split ratios don't really matter as much. At that scale, you'll likely be taking smaller samples anyway, so you can use any ratio that makes the data easier to work with. Now, there will be plenty of cases where you don't have a lot of data. Most of your homework assignments and student projects fit into this category. In industry, this can be common too. Often times it is expensive to get labeled data. Even if you're from a huge company, you still might have some problems where the data sets are fairly small. When we encounter this issue, we can use a method called k-fold cross-validation, which simply is a resampling method that uses different portions of the data to train and validate the model on different partitions of the data. This method allows us to recycle the data in a strategic way so that we can use every data point for both training and validation. K-fold cross-validation starts with first reserving a small subset of the original data to use as your test set, as we normally do. Then you take the remaining data and split it into k-equal size partitions, which we often call folds. We can depict this in the graphic shown here, and this can easily be implemented from scratch, but scikit-learn provides an easy class called k-fold, which makes this even easier. The next step is to iterate through each fold to

create a training and validation split. Within each iteration, we split the data, so the fold associated with the iteration number becomes the validation data. The remaining data is the training data. Within a given iteration, we then apply our standard procedure, which is to fit a model to the training data and evaluate it against the validation data. Each iteration will produce a loss from the validation set. Here we will be using whatever evaluation metric is appropriate. Once the loop is done, we would have k-separate measures, each taken from a separate partition of the data used as validation. Our validation loss is then just the average of this k-separate estimates. Although this method takes longer to train because we have to run the training procedure k-separate times, we get to use every data point for both the training and validation. It would seem like this is universally better, but with more sophisticated algorithms and a lot of data, each training step could take a while. Using this method will depend on the context, such as how much data you have and what your time budget is. Another important decision to make is how many folds to use. The most common number is to use 10 folds. This is supported by some theory we won't cover, but also has a lot of empirical validation. Just remember, though, that with more folds you'll need more time for training. Again, your time budget needs to be considered when doing this. Here is that whole procedure written out as actual code. You may want to pause this video a bit just to scan through the code and become familiar with it. For convenience, I have drawn red arrows to the specific lines that manage the k-fold process. We first define our faults using the k-fold classroom scikit-learn. We then loop through the procedures using the k-fold split method. All this split method does is create indexes for us that we can use to explicitly split the data into train and validation sets. We then estimate a model and get the validation loss. In this particular case, I'm using a validation metric called zero-one loss, which is also called accuracy. Finally, at the end, we summarize the procedure by taking the average of the validation losses. When we need to compare a different model configurations from our candidate set, this last statistic will be our main input used for comparisons.

Back to Table of Contents

Read: Out-of-Sample Validation Techniques

As we have discovered, splitting our data into different subsets can help us properly evaluate whether our model generalizes well to new data. After we've reserved data for a test set, there are different methods we can use to split our remaining data so as to train on some of the data and use the rest to validate our model's performance, then determine whether it is making accurate predictions on new data.

There are two common methods for splitting data into training and validation sets: the holdout method and the k-fold cross-validation method.

☆ **Key Points**

Two common methods for splitting data into training and validation sets are the holdout method and the k-fold cross-validation method.

The holdout method involves randomly splitting a data set into training and validation subsets.

In k-fold cross-validation, we split the data set into k subsets, training and validating k times such that each time we train on k - 1 training folds and test on one validation fold.

The **holdout method** is the simplest method; it involves randomly splitting a data set into training and validation subsets. We train on the training set and test our model's performance on the validation set. This method is typically used when we have a very large data set.

In the **k-fold cross-validation method**, we split the data set into equally sized subsets, or folds, with k representing the number of folds. We perform the holdout method (train and validation) k times, such that each time, we train on k - 1 training folds and test on one validation fold; in this way, every fold will have a chance to serve as a validation set. We then average the resulting prediction accuracies obtained on each of the k iterations to get a good estimate of our model's performance on new data.

Let's look at an example of five-fold cross-validation as illustrated in Figure 1:

1. We split the initial data set into five equally sized folds.
2. At each iteration, we train on the four green folds and validate on the one blue fold. In this example, we first train on Folds 2 through 5 then validate on Fold 1. We continue this training and validation process in a round-robin fashion until all five folds have had a chance to be the validation set.

3. We average the resulting prediction accuracies obtained on each of the five iterations to estimate how effective our model is.

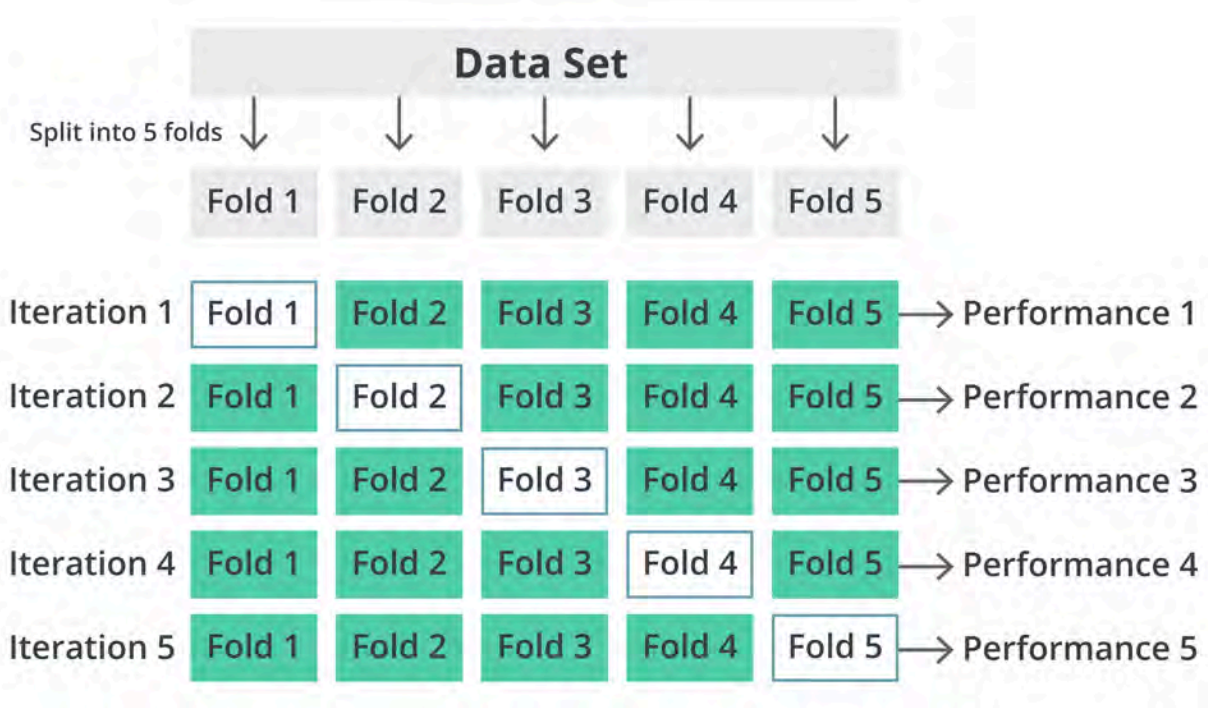


Figure 1. Example of five-fold cross-validation.

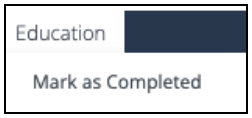
Back to Table of Contents

Assignment: Performing Cross-Validation for a KNN Model

In this exercise, you will use built-in cross-validation tools from scikit-learn to perform a k-fold cross-validation when training a KNN classifier.

This exercise 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. This assignment will be auto-graded and can be resubmitted. After submission, the Jupyter Notebook will remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

This exercise will be auto-graded.

Please complete this activity in the course.

Back to Table of Contents

Watch: Model Selection Demonstration

In this video, Mr. D'Alessandro demonstrates a full walk-through of the model selection process. You will see how to perform model selection on decision trees of various max depths to find the model that results in the best k-fold cross-validation accuracy score. Mr. D'Alessandro also introduces the idea of the grid search, which is offered by scikit-learn as a systematic approach to narrow in on the optimal model and its respective hyperparameters.

Note: You will see "AUC" referenced in this video. The AUC is a metric used to evaluate a model's performance. You will learn more about this evaluation metric later in this unit.

Video Transcript

In this lecture, we're going to provide a full walk-through of the model selection process using k-fold cross-validation. We'll use a real data set, though to be concise, I won't cover the details of that data. Just know that this is a supervised learning task. The first thing we'll do is choose our model candidates. We're going to keep it simple and just use decision tree classifiers as our sole algorithm. We'll consider one hyperparameter, max depth, to control the complexity, and we'll use all available features in the data. Here is a block of code that runs the model selection routine for us. At the top, we initialize the k-fold. Notice that we're setting a random state. This is a subtle but very important point. When we do model selection with cross-validation, we need to use the same full definitions for each model candidate to ensure our comparisons are on the same data. Since we're only searching over max depth parameter, I then define a range using increasing powers of two. After that, we get into the main loop. I first initialize the model candidate and then pass it into the cross-validation function. This function returns a metric. In this case, I am using a metric called AUC, which is a classification metric that we want to maximize. The main thing to know is, this is a metric that we want to increase again, and it has values between 0.5 and 1, where 1 is closer to a perfect model. After looping through all candidates, I store the result of each loop in a data frame. Here are the results of that procedure plotted. Notice how this plot shows the same shape that I showed in prior lectures when plotting model complexity versus expected loss. This time, though, the shape is inverted because I chose an evaluation metric we wish to maximize as opposed to minimize. In this case, trees with depth less than four are underfitting, so the AUC is between 0.58 and 0.6. The higher values of max depth are severely overfitting, so they tend to have the lowest AUC here. Again, the lowest AUC you should see is 0.5, which is equivalent to a model that is just randomly guessing. With that in mind, the scores on the right hand side are actually pretty bad. The best option for us is somewhere in

the middle, with max depths between 4 and 8 showing comparably good scores. I want to call out that scikit-learn has made this entire procedure possible in just a few lines of code. There is a module called grid search cv that performs cross-validated grid search for you. Grid search just means we specify a quote-unquote grid of hyperparameters, and we're searching through different combinations of them. This particular example is limited to a single algorithm, but it lets you search through a range again of hyperparameter options that you specify. As we should expect, this returns the same best value as what I produced with a more manually coded-up version. I'll leave it to you as an exercise to look at the documentation of this particular module and learn how to use it.

Back to Table of Contents

Read: Model Selection

Model selection is the process by which we choose one final model from among many candidates that we determine is the optimal model for a given data set and machine learning problem. We care about model performance as well as complexity. Choosing an optimal ML model involves three things:

1. Performing out-of-sample validation so we can properly evaluate how our model generalizes to new, previously unseen data. We split our data into training, validation, and test data sets. We use the training data for model fitting, the validation data to perform model selection (i.e., after evaluating the model's performance, we tweak the model's hyperparameter configurations accordingly), and a test set for the final evaluation of the chosen model's performance.
2. Performing feature selection to reduce the number of features, so as to only use relevant data in the training of our model. This, in turn, may improve a model's performance.
3. Determining the optimal hyperparameter configuration that results in a well-performing model.

☆ Key Points

The model selection process involves performing out-of-sample validation and feature selection as well as choosing the optimal hyperparameter configuration

Choosing the best hyperparameter values is an empirical process that generally employs systematic techniques to search for values

The two most commonly used searching techniques are grid search and random search

Choosing hyperparameter configurations

Let's focus on how we choose the optimal hyperparameter configuration for a given model. Choosing the best hyperparameter values is an empirical process. We often determine the optimal combination of hyperparameters through techniques such as grid search and random search. Using these techniques can be a time-consuming process, but it is usually worthwhile in order to pinpoint the optimal sets of hyperparameters for maximizing model performance.

A **grid search** essentially goes through various combinations of hyperparameters systematically, i.e., in geometric progression or exponential progression. This ensures good coverage within some regions determined by prior knowledge or heuristics.

Random search is like grid search, but instead of choosing our hyperparameter values systematically, we will select them at random in order to capture a wide variety of combinations that could otherwise be missed by being systematic about choosing our values.

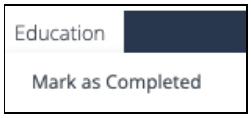
Back to Table of Contents

Assignment: Performing Model Selection to Choose a DT

In this exercise, you will perform decision tree model selection to find the hyperparameter configuration that results in the best cross-validation accuracy score. You will use different techniques to accomplish this, including your own implementation and built-in tools from scikit-learn.

This exercise 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. This assignment will be auto-graded and can be resubmitted. After submission, the Jupyter Notebook will remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

This exercise will be auto-graded.

Please complete this activity in the course.

Back to Table of Contents

Ask the Expert: Ask the Expert: Mehrnoosh Sameki on How to Choose the Best Model for Your ML Problem

In this video, Dr. Sameki discusses how to choose the best model for your problem. She explains the "no free lunch" theorem, which stipulates that no one machine learning model works best for every problem, and examines the different considerations to make when selecting the right model.

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.

How do you choose the best model for your problem?

Video Transcript

In machine learning, there is something called the no free lunch theorem, which means no one algorithm works well for every problem. As a result, you should try many different algorithms for your problem while using a holdout test set of data to evaluate performance and select a winner model. Some of the factors affecting the choice of a model are, one, whether the model meets the business goals, two, how much data pre-processing the model needs, three, how accurate the model is, you might want to use a variety of performance metrics to compare your model, four, how explainable or transparent the model is. Some models are more transparent and understandable than others depending on your use case, you can look through their architecture and see how they make their predictions, five, how fast the model is, meaning how long does it take to build a model, and how long does it take to make predictions? Six, how scalable the model is once it is deployed to production, meaning how efficiently it can handle high dimensional data with large number of features and thousands to billions of examples and instances.

Back to Table of Contents

Module Wrap-up: Perform Model Selection

In this module, Mr. D'Alessandro introduced an indispensable step in the model development process and explained the process of selecting the best model for a machine learning problem as well as the purpose of performing out-of-sample validation. You were also introduced to the three key principles for working with test data sets and various different sampling techniques, including k-fold cross-validation, which gives you the ability to work with small data sets. Finally, you were able to put the knowledge into practice by completing two practical coding assignments, one for performing cross-validation and the other for model selection.

Back to Table of Contents

Module Introduction: **Feature Selection**

Not all features are helpful for solving a machine learning problem, especially when computational cost is a consideration. Another important step in choosing the best-performing model is selecting the features that contribute to good performance and eliminating those that do not; this is called feature selection.

You have already considered feature selection when performing exploratory data analysis during data understanding and preparation, but there are specific feature selection methods commonly used in practice. In this module, Mr. D'Alessandro will introduce three common feature selection methods and explain the motivation and tradeoffs behind each method. To better solidify your knowledge on this topic, you will work on an assignment that provides hands-on experience with a feature selection method.

Back to Table of Contents

Watch: Feature Selection Overview

Feature selection is another key component of the model development process. In this video, Mr. D'Alessandro explains the motivation behind feature selection and introduces three commonly used feature selection methods: heuristic feature selection, stepwise feature selection, and regularization.

Video Transcript

In this lecture, we will start covering feature selection. Feature selection is the process of empirically testing different combinations of features to choose an appropriate set. The methods we learned for model selection apply to feature selection, but there are a few different approaches which we'll cover here. Before we learn how to do feature selection, let's first discuss why we do it. In an era of big data, a naive point of view might be to always leverage all of the data we have. Feature selection takes effort, after all, so we want to make sure the effort is justified, and usually it is. Here are some of the main reasons why. The first is that feature selection can help reduce overfitting. It is generally likely that some of the features we build have little to no predictive power, and as a general rule, we want to remove all of these features. Including such unpredicted features increases the chance of over fitting. Next, should we want to better understand our models, having fewer features makes the models easier to explain. The other two benefits are related. When you're a machine learning engineer, you need to always consider the scalability and stability of your prediction system. Every feature you use incurs runtime costs, development costs, and maintenance costs. If there is one motivation for feature selection that is difficult to refute, it's this. There are several approaches we can take for feature selection. I'm going to cover three of the most common methods here. Each of these represents different styles or classes of approaches at a high level. Within each there are variations to learn. Most feature selection is performed using model performance as a selection guide. The method I call heuristic selection here is the exception, which is why we use the term heuristic. With this method, you create some rule or heuristic to either sort or filter features prior to any machine learning. The second method, called stepwise selection, involves empirically testing different sets of features using the model selection techniques we learned in previous lectures. It is called stepwise because usually we do this in an incremental fashion. The last method, called regularization, is very unique from the rest. This method involves changing the loss functions of the algorithm to incur extra loss for each feature we use. With this approach, each feature needs to add material value to the model to compensate for the extra penalty.

Watch: Heuristic Feature Selection

The first of the three common feature selection methods is heuristic feature selection. Follow along as Mr. D'Alessandro delves into the motivation behind this approach with a real-world scenario using Twitter as an example. He introduces the idea of feature support and explains how it can be used in conjunction with stepwise selection, which will be further described in the next video.

Video Transcript

In this lecture, I'll present what we call a heuristic feature selection. This is a powerful method to use, particularly because it provides a great balance between scalability, simplicity, and efficacy. We call it heuristics selection because we're not going to use traditional model selection techniques as our guide. Instead, we design a rule before any modeling is done, and we use that rule as our basis for selection. Here are some common rules we might use to filter features prior to modeling. The first two that I'll present are data driven. Just because we are using heuristics doesn't mean we can't rely on data to make intelligent decisions here. The first rule itself is a kind of lite form of supervised learning. Here we are considering the relationship between the label and the feature. If we can't detect a relationship, we can remove the feature. This is probably the most effective rule amongst the said because it is directly informed by the label. As an example, we might use the correlation or information game between a feature and the label to rank and then select. The second rule considers the properties of the feature itself. Here I use a term called support, which is defined as the percent of examples where the feature is not 0 or null. The intuition is that 0 and missing values are cases that don't add predictive power to the model. When the support of a feature is low, we wouldn't expect the feature to perform well in the final model. This method, again is highly scalable, and is particularly useful when you're dealing with problems that have millions of features, which is fairly common in fields of computational advertising. The last rule is more generic. In the course of your work, you'll likely encounter different reasons why you would exclude a feature, independent of any data. As an engineer, you may find that the process that generates the feature is expensive or difficult to maintain. In some industries, specific features may be prohibited for regulatory reasons. This is a rule that you should probably consider first, as you probably shouldn't even invest efforts in building features you know you can't use. Another thing to remember is we don't need to commit to just one of these rules. As a matter of fact, you can apply all three in your feature selection process. I'll present one example here using the second rule regarding a feature support. When our data

comes in the form of text, one common approach to feature engineering is to convert each word to a binary feature. These features would be 1, if a particular word is in the document or the example, and 0 otherwise. Let's use a tweet as an example. Tweets are pretty short with few words, and there are a lot of tweets. The collection of millions of tweets will include a lot of words, probably in the order of magnitude of millions of them. But each tweet will only contain a few, and we might expect a lot of words to be pretty rare. To do feature selection here, we can start by computing the support of each word feature. The example chart here shows the support or the percent of times that a particular feature has a value of 1. If we sort the features and plot this value, we'd likely get a curve that looks like this. The right hand side is what we call the long tail. It is common in text data to have a small set of words that have medium to high support, with most words having low support, meaning, again, that they occur pretty rarely in the data. We can define a rule that selects all features above a given support level, which is depicted by the green shaded area. Of course, the challenge here is that I've introduced a decision parameter, which is the exact threshold we need to choose for feature inclusion. The red lines here show different thresholds we may have chosen for feature selection for our given application. We can decide this threshold by just picking one and sticking to it, or we can actually take an empirical learning approach. The threshold or the number of features that threshold yields can become a hyperparameter that we ultimately select using model selection techniques. This is where our heuristic selection process intersects with stepwise feature selection. We can set up an experiment where we start with the left most threshold and run cross-validation. Then we can iteratively increase the threshold and choose the value that gives us the best cross-validation performance.

Back to Table of Contents

Assignment: Performing Feature Selection Using Scikit-Learn

In this exercise, you will once again perform a grid search to find the optimal hyperparameter configuration that results in the best-performing decision tree classifier. You will then train and evaluate your model using the best hyperparameter settings and use scikit-learn's built-in metrics to determine the importance of each feature in creating the predictive model.

This exercise 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. This assignment will be auto-graded and can be resubmitted. After submission, the Jupyter Notebook will remain accessible in the first tabbed window of the exercise. To reattempt the work, you will first need to click **Education —> Mark as Uncompleted,** then proceed to make edits to the notebook. Once you are ready to resubmit, follow steps one and two.

This exercise will be auto-graded.

Please complete this activity in the course.

Back to Table of Contents

Watch: Stepwise Feature Selection

The second feature selection method is stepwise feature selection, which is a systematic approach for selecting features with the help of out-of-sample validation. In this video, Mr. D'Alessandro walks through the pseudocode of stepwise feature selection and explores the tradeoff between performance and cost.

Video Transcript

Now we'll cover stepwise feature selection. This is a straightforward but robust process because we can leverage model selection techniques and out-of-sample performance to inform our selection decisions. Here's the actual algorithm represented here as pseudocode. Let me walk through this step by step. The first step is to initialize our best subset, which at first is just an empty set. Then we initialize our candidate features, which initially is the set of all of the potential features we're exploring. After that, the main operation begins. We loop through each feature, and for each feature, we add it to the best subset and get a validation score. We can use either cross-validation or regular validation at this stage. When I say we are getting a validation score, what I mean is we train a model specifically with just the best subset at that time. We store all of these scores and then when we're finished with the loop, we choose the feature that produced the best validation score. Then we make two updates. First, we add the best performing feature to the best subset, and we also remove the best feature from the candidate set. Last, we repeat the loop in updating steps until some stopping criteria is met. One part of this algorithm that needs more explanation is the idea of a stopping criteria. The plot here shows an example output of the stepwise selection process. Each point represents the validation loss after training a model on the best subset up until that point. As a side note, sometimes we want to minimize this loss and other times we want to maximize it. It depends on the metric we're choosing. The important point now is we're minimizing some loss metric represented here. It's hard to tell exactly, but the best performing iteration is the last one. Now, I've highlighted a region that is the most interesting to me. We see really strong improvement over the first iterations, but once we get to iteration four, we start to get smaller and smaller incremental performance. The iteration marked 7 here, shows 99% of the best performance possible, which is measured with all the features. This represents the type of trade off we need to subjectively make when setting our stopping criteria. We can always just choose a fixed number of features and stop there but that's a naive approach that might miss the elbow we see here around step three. Instead, a good way to define a stopping criteria is to stop when the relative improvement after each iteration

is below some level. I've highlighted this as a formula on the plot. If we choose epsilon as, say, 1%, then we'd be stopping here around iteration 7. After seven features, we continue to get better performance but the incremental benefit is so low that it may not be worth incurring the extra future cost. The epsilon you choose is a subjective decision you'll have to make, but some low percentage improvement like 1% usually is a safe bet.

Back to Table of Contents

Watch: Using Regularization for Feature Selection

You already implemented regularization to reduce the complexity of a linear model. As a result of reducing complexity, regularization is essentially performing feature selection. In this video, Mr. D'Alessandro explains how regularization can be used to perform feature selection. He starts off by explaining the concept of implicit feature selection and how it relates to regularization. He also discusses which machine learning algorithms allow for regularization as a form of feature selection and which do not.

Mr. D'Alessandro also differentiates between the two types of regularization, known as L1 and L2. He lists the tradeoffs between L1 and L2 as well as the tradeoffs for the different sizes of parameter C. Note that for feature selection, L1 is the regularization method that is commonly used.

Video Transcript

In this video, I'm going to cover the process called regularization. Before explaining exactly how regularization works, let me first introduce the idea of implicit feature selection. With other feature selection methods, we have to explicitly run certain selection algorithms to reduce the feature count. Regularization, on the other hand, lets us reduce the future count within the model training process itself. You'll see that we still have to run model selection procedures to choose the relevant hyperparameters associated with regularization, but we wouldn't also have to run feature selection methods. Regularization performs this implicit feature selection by modifying the loss function we use for training. Note that because of the way this is constructed, we only use regularization with a specific set of learning algorithms, like logistic and linear regression. This method doesn't apply to decision trees or k-nearest neighbors. But I should add the caveat: we can use stepwise and heuristic feature selection within logistic regression, we just don't need to. Likewise, we would never use regularization with decision trees or k-nearest neighbors. As shown here, we redefine our loss function for logistic or linear regression by adding what we call a penalty. The penalty usually takes one of two forms that I'll describe in a bit. Intuitively, the penalty is related to the number of features you have in your model and their particular weights. So the more features you use, the larger the penalty becomes. This creates a tradeoff for you to manage. Since adding a feature causes a larger penalty, which always goes up, you'll need that feature to contribute to reducing the loss by an amount at least as high as that particular penalty. This tradeoff is modulated by the parameter C, particularly for implementation within scikit-learn logistic and linear regression classes. When C is large, the features incur less penalty, which means

they don't need to contribute a high reduction in the loss component to be included. The opposite is true when C is small. Now let's look at the penalties. First, we have two variants that we call L1 and L2. These are also called the lasso and ridge regression, respectively. In both cases, we're running a sum over the weight of each feature. The difference is that in L1 we take the absolute value of the weight and in L2 we take the square of the weight. Both of these methods cause the weights that the model learns to get smaller because larger weights incur higher penalties. Here is an illustration of the effect of regularization on a model's weights. In this plot I am showing the weights for two features on a data set where I performed logistic regression. I also use both L1 and L2 regularization methods. The features are represented by the line colors and the regularization type by the line format. In all cases, we can see the pattern I described. As C goes up, the regularization penalty goes down, which allows the feature weights to be larger. Notice how there tends to be an asymptote where increasing C doesn't really have any effect anymore. This is a fairly common phenomenon. On the left hand side, the weights are 0 or near 0 for several values of C. Because of the scale, the left hand data side is hard to see in full detail. The section highlighted in yellow is expanded in the bottom right chart. What I want to call out is that when we use L1 regularization, the feature weights tend to be exactly 0 when the penalty is high. With L2 we get small weights, but they're not usually exactly 0. This is ultimately where the implicit feature selection actually takes place. After running hyperparameter selection to identify the best value of C, we can review which features have weights 0 or something really close to 0. We can then remove these features from our data because the resulting model would no longer use them.

Back to Table of Contents

Module Wrap-up: **Feature Selection**

Feature selection is an important step in developing a viable machine learning model. In this module, Mr. D'Alessandro introduced the three feature selection methods commonly used in practice: heuristic feature selection, stepwise feature selection, and regularization. Heuristic feature selection applies nontraditional rules to selecting the best features, and it usually offers a great balance between scalability, simplicity, and efficacy. Stepwise feature selection is a systematic approach to building up a set of features until model performance no longer improves. Regularization is a technique used to reduce overfitting, which can also be used to eliminate insignificant features. Now that you've completed this module, you will be able to put these feature selection methods into practice.

Back to Table of Contents

Module Introduction: **Choose Model Evaluation Metrics**

It would be impossible to choose the right model without having some objective quantitative metrics to help us evaluate how well a model performs. Depending on the type of problem we are trying to solve, the evaluation metrics used to quantify a model's performance will be different.

In this module, Mr. D'Alessandro will define the various evaluation metrics that can be used to understand a model's performance.

Back to Table of Contents

Watch: Confusion Matrix and Classification Metrics

Many of the evaluation metrics for classification derive from a concept called the confusion matrix. In this video, Mr. D'Alessandro examines what a confusion matrix is and how it forms the basis for three common classification evaluation metrics: accuracy, precision, and recall. Mr. D'Alessandro further defines each metric and explains when to apply them. He also introduces the idea of base rate, which is an important aspect to consider when evaluating a model's performance.

Video Transcript

Many metrics and concepts related to classification measurement derive from a simple table called the confusion matrix. In this lecture, I'll walk through how the confusion matrix is derived from a model's prediction and then cover key components and metrics we can derive from it. Let's first start with a binary supervised learning model that produces a probability score, like a logistic regression or a decision tree. As shown on the left hand side here, we assign a particular example to a class by determining which side of a given threshold the example score lies. When we run this process on labeled data, we have four different scenarios that will be created. These scenarios are represented on the right here by what is called the confusion matrix. When the prediction matches the outcome, we have either a true positive or a true negative. If we predict a positive label, but the actual value is negative, we call that a false positive, and the reverse of that is a false negative. When building a confusion matrix, we would fill in the cells with the accounts observed from our evaluation data set. We can then use these counts to compute different classification metrics. The three most common classification metrics will derive from the confusion matrix are accuracy, precision, and recall. Each of these is represented three ways in the illustration provided here. In each case, I highlight which components of the confusion matrix are used to compute the particular metrics. I use green highlights for the elements in the numerator and red for the elements that are added to green to make a denominator. We can think of each metric as a percentage or rate that captures different concepts. Accuracy is the most traditional metric. That represents the percent of all classifications that match the actual value. This is also called zero-one loss. Precision focuses just on the positive predictions and tells us what percent of those were actually classified correctly. Recall focuses on the actual positive cases, as in the true labeled cases, and tells us what percent of the actual positives were classified as positives. So now we have three metrics for classification problems, a natural question is when should we use each one? Technically, one can always measure all of

them, but when it comes to procedures like cross-validation and model selection, we usually need to make decisions based on just a single metric. Here are some common patterns that help define when to use each one. First, accuracy is actually not commonly used. In many binary classification problems, we usually want to input emphasis on the positive cases because these are the cases that we'll usually act upon. If it's a binary classification problem where false positives are more expensive to us than false negatives, we'll emphasize precision. In the opposite case, will emphasize recall. In multi-class classification problems, precision and recall aren't really defined because it really is about binary classification. In that case, we can use accuracy by default. Another very important rule to follow when using these specific metrics is to always use a baseline to compare your models. The reason is the idea of what we call base rate and variance. The base rate of the model is simply the percent of cases in your evaluation data where Y equals 1, or it's the average value of Y assuming it's coded up as 1 and 0. When we look at a specific measure of accuracy, precision or recall, the value we put on that specific measure depends on really what the base rate is. Here are two example problems. Let's assume they represent the accuracy of models built on different data sets. In problem one, 10% of labels are 1, and problem two, the same base rate is 50%. Now, if I report that the accuracy of problem one is 90%, is that a good or a bad result? Think about what the accuracy would be if we predicted negative or 0 for all examples in either problem. In problem one, since 90% of the examples are truly negative, our accuracy would actually be 90%. In this case, the accuracy you see here is what you get if you just made random guesses or assign the same label to every single case. This is technically high accuracy, but it also wouldn't be a very useful model. Now in problem two, the same strategy would yield an accuracy of 50%. But here I report 70%, which is a big improvement of the case of random guessing. The advice here is to never report accuracy or precision and recall similarly without comparing it to a baseline result. The simplest baseline is to consider the result you would get from randomly guessing or guessing the same output for each example. As you iterate, your baseline can be the most recent best performing model. In either case, the emphasis should be on the relative improvement over the baseline, not necessarily the absolute number of the metric. Again, this rule applies to all three metrics discussed in this lecture.

Back to Table of Contents

Read: Classification Evaluation Metrics

The goal of training a machine learning model is to generalize to new, previously unseen data. While performing well against training data is a good starting point, we need to make sure that the model also generalizes well against new data.

We need some kind of objective way of measuring the performance of our model, and this is where evaluation metrics come in. Depending on the nature of the problem (e.g., regression or classification), different evaluation metrics are used. Let's focus on classification metrics.

☆ **Key Points**

Accuracy refers to an overall match between predicted outcomes and actual outcomes.

Precision refers to how well a model can predict key outcomes.

Recall refers to how many key outcomes were predicted.

Imagine you have built a binary classifier that predicts whether a customer will make a specific purchase. The model is predicting one of two class labels: "Buy" or "Not Buy." We can determine how well the model performs by applying it to the validation set. Examine the confusion matrix for the binary classification purchase prediction model below (note that a confusion matrix can be used in multiclass classification as well).

Example of the confusion matrix for the binary classification model.

Confusion Matrix

		Actual Values	
		Number of True Positives (TP): **Model correctly predicts a purchase.**	Number of False Positives (FP): **Model incorrectly predicts a purchase.**
Predicted Values		Number of False Negatives (FN): **Model incorrectly predicts a non-purchase.**	Number of True Negatives (TN): **Model correctly predicts a non-purchase.**

Confusion Matrix for a Purchase Prediction Model

		Actual Values: Positive (Buy)	Actual Values: Negative (Not Buy)	Totals
Predicted Values	Positive (Buy)	TP = 15	FP = 10	25
	Negative (Not Buy)	FN = 5	TN = 20	25
	Totals	20	30	50

Using the confusion matrix as a reference, let's see how three evaluation metrics can predict the performance of our classification model. We will focus on accuracy, precision, and recall.

Accuracy

The most common and intuitive example of an evaluation metric is **accuracy**. You have been using the accuracy metric to evaluate your classification models. The accuracy of a model is simply the number of correct predictions (TP + TN) divided by the total number of predictions (TP + TN + FP + FN):

$$Model\ Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

In the case of our purchase prediction model, the number of correct predictions is 35 (which is 15 + 20 on the diagonal) over 50 total predictions, for 70% accuracy.

$$Model\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{15 + 20}{15 + 20 + 10 + 5} = 0.70$$

Sometimes, however, accuracy may overstate the fit of a model, as we need to account for the accuracy of guessing, or simply being correct by chance. Our model "Buy"s could have matched the actual "Buys" by chance alone, with a probability of 25/50 * 20/50 = 0.2. Similarly, the "Not Buy"s could have matched with probability 25/50 * 30/50 = 0.3. Therefore, simply by chance, we could have had an accuracy of 0.5; i.e., 0.2 + 0.3. A measure called **kappa** is used to adjust our model accuracy by this random accuracy, as follows:

$$Kappa = \frac{Model\ Accuracy - Random\ Accuracy}{1 - Random\ Accuracy} = \frac{0.7 - 0.5}{1 - 0.5} = 0.4$$

Precision

The **precision** of a model refers to how well the model can correctly predict a specific outcome; it is a measure of quality. Precision is calculated as the proportion of the positive predictions that were correct. If we are focused on predicting purchases, precision is the number of correct purchase predictions (TP) over the total number of purchase predictions (TP + FP).

$$Model\ Precision = \frac{Number\ of\ correct\ purchase\ predictions}{Total\ number\ of\ purchase\ predictions}$$

Our model predicts 25 purchases (which here is the sum of 15 + 10) against 15 of those actually purchased, so it yields a **precision** of 15/25, or 60%.

$$Model\ Precision = \frac{TP}{TP + FP} = \frac{15}{15 + 10} = 0.60$$

Recall

Recall is concerned with the number of times the model correctly predicted a specific outcome. While precision focuses on quality, recall focuses on quantity. Recall calculates the proportion of actual positive classes that were predicted; it is also known as the true-positive rate.

If we are focused on predicting purchases, recall is the number of correct purchase predictions (TP) over the total number of actual purchases (TP + FN).

$$Model\ Recall = \frac{Number\ of\ correct\ purchase\ predictions}{Total\ number\ of\ actual\ purchases}$$

There were 20 (15 + 5) actual purchases. The model correctly predicts 15 of those purchases; therefore, the model yields a **recall** of 15/20, or 75%.

$$\textit{Model Recall} = \frac{TP}{TP + FN} = \frac{15}{15 + 5} = 0.75$$

Improving recall typically reduces precision and vice versa. To properly evaluate your model, you must consider both precision and recall then find the optimal balance between the two.

Back to Table of Contents

Code: Confusion Matrix Demo

In this activity, you will explore how to implement a confusion matrix using scikit-learn.

This activity will not be graded.

Please complete this activity in the course.

Back to Table of Contents

Read: Choosing a Classification Threshold

You learned about choosing a classification threshold when working with logistic regression. Recall that the output of a logistic regression model is a probability that an example belongs to a given class. Logistic regression is just one of a handful of machine learning algorithms that produce probability estimates. To map these probabilities back to class labels, we have to choose a threshold.

☆ Key Points

Classification thresholds are used to map a probabilistic model's output to a class label.

Thresholds are problem dependent and can be adjusted and tested to improve a model's performance.

Precision and recall evaluation metrics can be used to help us choose a classification threshold.

While it is tempting for us to define a threshold to be 0.5, it is not always best practice to do so. Thresholds are problem dependent and therefore values that can be tweaked to improve the performance of a classification model. Let's consider some examples of how precision and recall can help us determine the best threshold.

Recall

Imagine we are a car manufacturer trying to predict the likelihood of having faulty brakes in our cars. It is important that all cars with potential faulty brakes get recalled since any false negative can lead to a disastrous outcome. Assuming a label of 1 means faulty brakes and a label of 0 means good brakes, we want to set our threshold very low — perhaps ~0.01 — so that even cars with the slightest suspicion of having faulty brakes get recalled.

Fittingly, the *recall* metric can help us determine a threshold for this model. Remember that recall is mathematically defined as follows:

$$\text{Recall} = \frac{\text{Number of True Positives}}{\text{Number of True Positives} + \text{Number of False Negatives}}$$

A low threshold will lead to a high recall value, capturing as many positive cases (1s) as possible. If our goal is to just capture a few of the most likely positive cases, then we can set the threshold pretty high.

Precision

Precision can help us choose the best threshold as well. Recall that precision measures how many of the positive predictions are correct and is mathematically defined as follows:

$$\text{Precision} = \frac{\text{Number of True Positives}}{\text{Number of True Positives} + \text{Number of False Positives}}$$

To illustrate where precision is important in determining a threshold is to think of the criminal justice system in the United States, where a criminal suspect is always considered innocent until proven guilty. We want to ensure that those convicted of a crime are, in fact, criminals. We don't want to risk putting innocent people in jail, so we have to reduce the number of false-positive guilty individuals. By setting the threshold to the right value, we will be able to ensure precision is maximized.

Different tools are available to help us understand a model's precision-recall tradeoff for various threshold values and determine the optimal threshold to choose. These tools that you will explore are AUC-ROC and the precision-recall curve.

Determining a threshold based on your machine learning problem and measuring the performance of your probabilistic model across many possible probability thresholds can enable you to choose the appropriate threshold that results in the best model performance.

Back to Table of Contents

Watch: Evaluating a Classifier Using the AUC-ROC Curve

The receiver operating characteristic (ROC) curve is used to visualize the performance of a binary classifier. It plots the false-positive rate against the true-positive rate (recall) at various classification thresholds. The area under the curve (AUC) is a metric that captures the performance of a classifier across various thresholds. It can also be used as a ranking metric that measures the ability of a model to distinguish between binary classes. The AUC compares the relative ranking of the probability estimates.

For example, if a classifier produces probabilities of 0.90, 0.70, 0.65, and 0.60 for the labels 1, 1, 0, and 0, respectively, this classifier is considered to have a good ranking performance since the actual labels for higher scores are 1s and lower scores are 0s. If the classifier produces probabilities of 0.90, 0.70, 0.65, and 0.60 for the labels 0, 1, 0, and 1, then it would be considered as having poor ranking performance. In this video, Mr. D'Alessandro explains how the AUC evaluates a model's ranking ability, how to calculate AUC from ROC, and the intuition behind them.

Video Transcript

Now we'll discuss a commonly used metric for a machine learning task called ranking. Ranking is a process where we use a model to sort items. When a classifier is used to rank, we can think of this as sorting items by the likelihood of the label being true. Ranking is how newsfeeds are delivered and is often used in marketing tasks amongst others. This metric we'll cover is the area under the receiver operator curve, or AUC for short. This metric is ideal for ranking problems, but is also suitable for any classification task that produces probabilities as predictions. I see it used more often in practice than other classification metrics. I'll show how this metric is constructed from the confusion matrix and provide tips on its usage. Common classification metrics derive from the confusion matrix and are tied to a specific threshold used to classify your examples. There are times though where we don't want to commit to a single threshold. These are if we're not sure about the relative cost of false positives and false negatives. If we're uncertain about our prediction budget and if our problem goal is to rank items. Let me explain what I mean by prediction budget here. In some problems, we might know exactly how many instances we want to classify. This is common in marketing where we have a specific dollar budget. From that we know we can send out a fixed number of ads, or in some recommender systems, we might always recommend a fixed number of items based on the design of the webpage. In other instances, this might not be the case. Take again, a newsfeed where the number of items

recommended is different for each person and maybe each person and each session. In this case, we wouldn't use a single threshold for classification. This is a classic ranking example where AUC would be the most appropriate metric. Again, beyond ranking, AUC is generally a great default metric to use for other classification problems. To construct the AUC, we go through an iterative process. Each of these vertical lines represents the same ranked list of classifier predictions. An important point here is the model needs to output a probability, not a binary output. To compute the AUC, we first build a curve called the receiver operator curve. We start by iterating through every possible threshold in the data, like as shown here. With each threshold, we compute the false positive rate and true positive rate from the confusion matrix and plot the points. If we connect these points with a curve, we end up with what is called the receiver operator curve, or ROC for short. This curve represents the false positive and false negative tradeoffs we would have to make for all possible classification thresholds. We can even think of this in terms of cost versus benefit. The y-axis is the true positive rate, which is another way of saying recall. We can think of this curve as telling us that to get to a certain recall, we need to tolerate a certain level of false positives. On this chart, I also show two dotted lines. These represent different sets of tradeoffs. The straight line on the bottom represents the worst-case scenario, which is when your model predictions are as good as randomly guessing. The line on top represents a better set of tradeoffs. When comparing the top to the other lines, we can see that for any given false positive rate, we can get a higher recall. Better models will always have an ROC that is steeper and higher on the left. We can capture the average quality of these tradeoffs in a single metric by taking the area under this curve. The AUC of this example is represented by the gray area on the plot. I have also drawn two important reference lines to consider. As I previously noted, the worst-case scenario is a model that performs like a random number generator. This ROC would be a straight line. And because the x and y axis have the range of 0-1, the AUC for the random baseline is 0.5. The other extreme is the perfect model. This model ranks all positive instances above the negative instances, so we can get perfect recall with no false positives, this model would have an AUC of one. Finally, as I call out here, the AUC has a nice intuitive interpretation, which is it represents the probability that a randomly chosen positive example from your data will have a higher model score than a randomly chosen negative instance. When we see an AUC of 0.5, the AUC represents a random coin flip. The ROC and AUC are useful tools for understanding the performance of binary classification models. It is fairly common to plot different model candidates against each other so that we can visualize the relative performances. This plot shows four different models against each other. We can see here that the shape of the curve is different for different values of AUC. When doing automated model selection, we can do this by just comparing the AUC. But for presenting model results to your peers, it is often helpful to show the full curves. The AUC is ideal for

ranking problems, but again, is useful for other classification problems you are working on. If you're unsure of what is the optimal threshold for your problem or you're decidedly doing a ranking problem and you don't want to commit to a specific false and true positive rate, the AUC is a safe metric to use.

Back to Table of Contents

Read: AUC-ROC Curve

The Area Under the Receiver Operating Characteristics curve (AUC-ROC) is a quality metric that can be used to compare two different binary classification models, assuming those models use probabilities to make predictions. A good example of this kind of model is logistic regression. We will start by recalling how logistic regression uses the probabilities it generates to make predictions by default and examine what happens if we modify the default behavior. Then we will look at how this information can be plotted into a ROC curve, and finally converted into an overall AUC-ROC score. We can then use AUC-ROC scores as one way to compare machine learning models built on different data.

Back to Table of Contents

Ask the Expert: Ask the Expert: Kathy Xu on Probabilistic Models

You've been introduced to a few evaluation techniques that are suited for classification models that output predicted probabilities. Such models are known as probabilistic models. Logistic regression is an example of a probabilistic model. In these videos, Ms. Xu discusses when to use a probabilistic model and offers an example of when she used one in practice.

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

Kathy Xu

Analytics Extensibility Lead, Pfizer

Kathy (Qingyu) Xu leads a team responsible for extending analytics capabilities across Pfizer. Her team helps create machine learning-powered products (web apps, plugins, and dashboards) for use across the enterprise and provides guidance for fellow data scientists and analysts to optimize their usage of data and analytical technologies that are a part of the enterprise analytics platform.

In particular, Ms. Xu has an interest in MLOPs and industrializing machine learning models. Outside of work, she is an avid art history lover and frequently visits the museums around NYC, such as the Cooper Hewitt and Brooklyn Museum. Ms. Xu received her Master and Bachelor of Science in Statistics from Cornell University.

Can you describe a project when you used a probabilistic model?

Video Transcript

In the past, we've leveraged probabilistic models to help create an algorithm to recommend to healthcare providers who is more likely to have a rare disease or condition based on different probabilities of an individual having other comorbidities. So it's really powerful for probabilistic models is that they're really helpful in cases where there's some uncertainty or variability in the data. In our particular use case here, we actually had to create different probabilities for different comorbidities. So in this case, for that particular rare disease, we had to figure out what is the probability of someone having another comorbidity such as heart palpitations that could affect this particular rare disease. So that was really helpful for us.

How do you know when to use a probabilistic model?

Video Transcript

One way that we can think about probabilistic models is that they're particularly helpful in situations where there's a lot of uncertainty or variability in the data. It's really important to understand the amount of uncertainty in those predictions as well. There a lot of times used in certain capabilities such as natural language processing, computer vision, speech recognition and so on and so forth. I think that maybe the three main, I'll say building blocks, of probabilistic modeling are: one, trying to figure out what is the adequate probability distribution. A lot of times we're looking at probability distributions or if you're looking at Bayes Theorem, where something falls within a distribution. The second thing I would take a look at is make sure you're using the correct use of input information for those different types of distribution functions. Of course, trying to understand the proper linkage between the interactions of all those different types of variables. One potentially tangible way you could use a probabilistic model is you could try to use a probabilistic model to predict, for example, how bad traffic might be given, let's say weather data. If it's for example snowing, sleeting, really bad icy conditions, you can create different types of probabilities to understand how bad the roads will be outside depending on those probabilities.

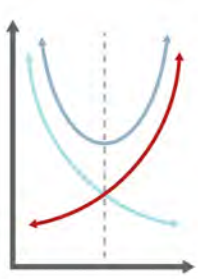
Back to Table of Contents

Module Wrap-up: Choose Model Evaluation Metrics

In this module, Mr. D'Alessandro introduced different metrics for evaluating your model's performance, including three important metrics specifically for classification; these metrics, derived from the confusion matrix, are accuracy, precision, and recall. You also discovered different methods to evaluate classification thresholds: the precision-recall curve and the AUC-ROC curve. This prepared you for the exercises where you built your own ROC curve and tuned a model.

Back to Table of Contents

Module Introduction: Deploy Your Model



Once you have trained and tested your model, how do you make it available to stakeholders to solve business problems? You have to deploy and host your model in a production environment. So how do you do that? And what happens after? In this module, you will explore different ways of deploying your model as well as things to monitor once your model is in production.

Back to Table of Contents

Code: Making Your Model Persistent

In this activity, you will see how to make your scikit-learn model persistent so that you can use it in the future to make predictions on new, unseen data without retraining your model every time.

This activity will not be graded.

Please complete this activity in the course.

Back to Table of Contents

Ask the Expert: Ask the Expert: Kathy Xu on Using ML Models in Production

Once a model has been trained and tested, it can be used in a production environment to solve business problems. In this video, Ms. Xu discusses the different ways ML models are used in production.

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

Kathy Xu

Analytics Extensibility Lead, Pfizer

Kathy (Qingyu) Xu leads a team responsible for extending analytics capabilities across Pfizer. Her team helps create machine learning-powered products (web apps, plugins, and dashboards) for use across the enterprise and provides guidance for fellow data scientists and analysts to optimize their usage of data and analytical technologies that are a part of the enterprise analytics platform.

In particular, Ms. Xu has an interest in MLOPs and industrializing machine learning models. Outside of work, she is an avid art history lover and frequently visits the museums around NYC, such as the Cooper Hewitt and Brooklyn Museum. Ms. Xu received her Master and Bachelor of Science in Statistics from Cornell University.

How to use the model in production?

Video Transcript

I think one of the main challenges that a lot of times we face, especially in the field of data science and data analysis, is once a model is created, how can we actually get that model

from the development process into production? There's a lot of different things we can think about. I think one of the main things to think about is understand where's is our data coming from and is there some sort of refresh schedule? Depending on where your data is coming from and the refresh schedule and how often a end user might expect, say, a refresh of the model or new output, you might consider building your model in different ways. What I mean by that is if you have a model that only needs to be retrained once a year, then it might just be enough for a data scientist or a data analyst, host this model in a particular environment, generate some sort of output, and then send that recommendation to a stakeholder like once a year through like a PowerPoint; I think that might be sufficient. However, if you're looking at a use case where an end user needs to see daily results or a daily refresh or something that's much more frequent than, say, a yearly refresh or a yearly readout, then what I would recommend individuals consider in that creation, in that process from going from development into production, is how do you want to display the end results to you at your end user? A lot of times what our team does is consider, "Hey, maybe we should create some web application or visualization for our end user and then create a model that's hosted in our environment so that it can be automated and refreshed at that" — you know, daily, weekly, or whatever cadence is available. Once that's completed, the actual output of that model gets read into our web application or into our visualization, and then the end user actually sees a live dashboard, for example, with a BI tool, whether that's using Tableau, Spotfire, Power BI, or a visualization that's created through, say, Dash Plotly and other methods, and the end user can actually visualize different types of plots and charts and output directly. So instead of going, say, through like a PowerPoint or the output of your Jupyter Notebook, your model's actually packaged within an environment, automated, and then the end user doesn't even see that part; they only see the end visualization with the cards and the end results that are fed through the visualization. So it's a little bit more — I'll say tangible, especially for users that maybe don't come from a technical background, and they're able to digest that information very readily.

Back to Table of Contents

Read: Deploying, Hosting, and Monitoring Your Model

You just practiced one way to make your trained and tested machine learning model persistent so that you can use it in the future to make predictions on new, unseen data. But in the real world, what do you have to do to make your model available to stakeholders to help them solve business problems? You have to deploy and host your model in a production environment. Deploying a machine learning model can be challenging, but advancements in cloud computing have made the process much easier.

We will outline the considerations for deploying, hosting, and monitoring machine learning models to ensure their ongoing success.

1. Determine the deployment method.

After the model is fully trained, it can be deployed to make predictions on new data through two primary methods: batch inference (or offline inference) and online inference (also known as real time). Batch deployment processes large volumes of data in batches on a recurring schedule and stores the predictions in a database. This information can be provided to stakeholders when needed. Online inference processes data as it arrives in real time.

The choice between the two methods depends on the specific problem requirements, and both come with their own set of advantages and disadvantages.

Deployment Method	Pros	Cons
Batch inference	• Can deploy more complex models with a larger number of inputs • Requires simpler infrastructure requirements and less computational power compared to online inference • Predictions can be analyzed and processed before being seen by stakeholders	• Can result in higher processing latency • Not suitable for applications that require real-time predictions, therefore predictions may not be available for new data
Online inference	• Provides results in real time and on demand • Lower processing latency	• Harder to implement • Cannot handle as complex models as batch inference • More computationally demanding compared to batch inference

Key considerations:

- Your latency requirements
- The complexity of your model
- Specific requirements of your use case

Questions to ask:

- How frequently do you require predictions?
- Do you need results based on individual cases or batches of data?
- What amount of computational power is needed to process the inputs?

2. Choose a hosting environment.

When deploying a machine learning model, one important consideration is where to host it. There are two main options — internal hosting or cloud services — and the decision often depends on the specific needs of the project and organization. Below are some key considerations when choosing where to host your model.

Deployment Method	Pros	Cons
Internal	• Greater control and security for sensitive data • May be more cost effective for larger companies with existing internal infrastructure	• Can be more costly in resources, time, and maintenance • More difficult to scale
Cloud services	• Easily scalable and flexible, with lower maintenance • More cost effective for smaller companies without existing internal infrastructure	• May not be cost effective on large projects • May not meet strict security requirements

Questions to ask:

- **Infrastructure:** Does your organization have the necessary hardware, network, and security protocols to support internal hosting? If not, consider cloud services.
- **Cost:** How much will it cost to host and deploy the model? Internal hosting may require a significant upfront investment, while web services are often charged based on usage.
- **Scalability:** Will your project grow in terms of data volume and user demand? Choose a host that can easily scale up to meet those needs, such as cloud-based web services.
- **Security:** Does your project have specific, strict security requirements? If so, deploying the model on premises can provide better control over the infrastructure and data.

Popular cloud services for deploying and hosting machine learning models

Deploying ML models to the cloud is an increasingly common practice, as it provides easy access to necessary computing power, storage, and network resources for handling large data volumes and running complex models. Yet there are also potential downsides to consider when weighing your options, including security concerns, high cost, latency, and dependency on the cloud provider.

- **Amazon SageMaker** is a fully managed machine learning service offered by Amazon Web Services (AWS). It provides a complete platform for building, training, and deploying machine learning models.
- **Google Cloud AI Platform** is a suite of tools and services that help you build, train, and deploy machine learning models on Google Cloud.
- **Microsoft Azure Machine Learning** is a cloud-based machine learning platform that enables you to build, deploy, and manage machine learning models.
- **BentoML** is an open-source platform for deploying, managing, and serving machine learning models. It provides a unified interface for packaging and deploying models as production-ready web services
- **Kubeflow** is an open-source platform for deploying and managing machine learning workflows on Kubernetes. It provides a unified interface for managing machine learning pipelines.
- **TensorFlow Serving** is a framework for serving machine learning models using TensorFlow. It provides a flexible architecture for deploying models in production.

3. Deploy your model.

Once you've chosen the deployment method and hosting environment that suits your project, the next step is to package the model along with its dependencies into a deployable format such as a container or a bundle. Containers are popular because they're predictable, reproducible, and easily modifiable, making them ideal for collaboration among engineers.

Industry-standard tools that specialize in preparing for deployment:

- **Docker** is a popular tool for packaging and deploying applications in containers. It can be used to package machine learning models and their dependencies into a container that can be easily deployed to different environments.
- **Kubernetes** is an open-source container orchestration platform that can be used to manage and scale containerized applications. It can be used to deploy and manage machine learning models packaged in containers.

- **ONNX Runtime** is an open-source runtime engine for deploying models that are compliant with the Open Neural Network Exchange (ONNX) format. It provides high-performance execution of models across different hardware platforms.

Before deploying the packaged model, you should test it to ensure that it performs as expected. This may involve running tests to evaluate the model's accuracy, precision, recall, and other performance metrics. Once you are satisfied with the results of your testing, deploy the packaged model to your target environment that has been determined based on your security, financial, performance, and computational requirements.

Automating deployment

To streamline deployment and testing workflows, some organizations automate the process. Automation ensures the model is tested regularly to maintain its robustness. It can also help scale the model without burdening the team.

4. Monitor for model improvements.

Monitoring a deployed machine learning model is vital to ensure it is performing well and to detect any issues that may arise during production. Monitoring the model's performance, identifying errors, and making necessary adjustments is crucial. Things to monitor for include:

- **Performance deterioration:** Quality degrades over time due to changes in data distribution.
- **Bias or discrimination:** This occurs when the data used to train the model is not representative of the population it is intended to serve.
- **Security risks:** Over time, attackers may be able to manipulate or alter the model's predictions to achieve their goals.
- **Costly revisions:** A model may require costly revisions or even replacement if its performance degrades over time.

Best practices for monitoring model performance

- Constantly evaluate your ML model's performance on real-world data to detect any decrease in accuracy due to changes in the data environment, known as model drift. If model drift occurs, retrain your model with fresh data to improve its accuracy.

- Monitor your deployment pipeline to ensure the model runs smoothly and debug it when necessary.
- Collect feedback from end users to improve the model's performance by identifying areas for improvement and providing valuable insights.

Considering all of the above factors and being methodical when deploying a machine learning model is important to ensure that the model is accurate and reliable and that it delivers the expected value to the business or project for which it is intended.

Back to Table of Contents

Tool: Deploying, Hosting, and Monitoring Your Model

Once you have trained and tested your machine learning model, you have to deploy and host your model in a production environment so as to make your model available to stakeholders to be used to solve business problems.

Download the Tool

Use **Deploying, Hosting, and Monitoring Your Model** tool as a guide to deploy, host, and monitor future models.

This reference tool outlines the considerations for deploying, hosting, and monitoring machine learning models to ensure their ongoing success. Use it as a guide when you are prepared to deploy your machine learning model.

Back to Table of Contents

Ask the Expert: Ask the Expert: Francesca Lazzeri on Deploying to the Cloud

In this series of videos, Francesca Lazzeri describes what the deployment of a machine learning model entails. She explains the process from algorithm development to application, going through the various deployment methods, expected time frame, and where to anticipate errors. Finally, she describes the different tools, platforms, and architectures most commonly used in the industry.

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

Francesca Lazzeri

Curriculum Committee Industry Advisor, Microsoft

Dr. Francesca Lazzeri is an experienced scientist and machine learning practitioner with both academic and industry experience as an Adjunct Professor of AI and Machine Learning at Columbia University and Principal Cloud Advocate Manager at Microsoft. She also authored the book "Machine Learning for Time Series Forecasting with Python" (Wiley) and many other publications, including technology journals and conferences. At Microsoft she leads a large international team (across the U.S., Canada, U.K., and Russia) of engineers and cloud AI developer advocates, managing a large portfolio of customers in the research and academic sectors and building intelligent automated solutions on the cloud. Before joining Microsoft, Dr. Lazzeri was a research fellow at Harvard University in the Technology and Operations Management Unit. You can find her on Twitter at @frlazzeri.

What does it mean to "deploy your machine learning model"?

Video Transcript

So, machine learning model deployment is simply the process by which a machine learning algorithm is converted into what we call the web service and then into an application, and as a data scientist, as a machine learning predictioner, so we refer to this conversion process as the operationalization, or also deployment of the machine learning algorithm. Operationalizing a machine learning model really means to transform it into something that is a consumable, so into a consumable service, we said, and as a consequence, then other people can embed it into an existing production environment. So it's really the moment in which you basically transform the model that you build into an AI application. Because thanks to this deployment process, then other people are going to be able to call your model and produce results out of it. So I really like to refer to it as really the moment in which machine learning becomes AI, becomes artificial intelligence.

What are the different tools, platforms, and architectures used in the industry?

Video Transcript

In the industry, I would say in general, not just for model deployment, but in general, the main tools that are used for building the machine learning models, training them, and testing them, and then of course, deploying them are like Python is a programming language that's been used a lot, as I said, not just for training, preparing your data, and testing your models, but also for deployment. It is great because, as you know, Python is an open source tool, so it's really supported by the community. There's always this sort of improvement because, as you know, Python has multiple versions, and there are a lot of learnings that we can observe from one version to the other, a lot of learning, and as a consequence, a lot of improvements. So, Python for sure is a tool, and the other, in my opinion, big tool, of course is cloud computing in general. I know that cloud computing can sound like a very broad term. But when I say cloud computing, I mean different typologies of services, such as computer targets that you need in order to attach your experimentation environment to a computer that then you are going to use in order to run your models. But also you have to think in terms of the storage. You need to access different storages in order to store your data, making sure that you have a working end to end data pipeline, meaning that you need to get new and refreshed data at specific internals. Then I think that the other important piece of all of these is also the data visualization. Data visualization is another tool that most of the time, I like to tell to my team of data scientists that they need to think about data visualization as an opportunity to understand the data, so at the beginning of the machine learning development process, but also at the end because data visualization is an extremely important tool that we have in order to communicate our results. Data visualization, again, we have a lot of different tools that we can think about Power BI, we can think about also Tableau. Also, Python is another great resource for data visualization with their Matplotlib package. I would say always think about not just one single component of the machine learning development process, but think about the end to end process. Then if I have to mention some of the important tools that I have seen the industry use, I would say again, Python, cloud computing, different computer targets, and different tools for data storage, and different tools for data visualizations.

How do you deploy a model?

Note: *The functions listed in this video are* `init( )` *and* `run(input_data)`.

Video Transcript

There are many different technologies and tools that you can use. Most of the time we like to say that there are two main functions that you need to use. Most of the time these functions can be written in Python. We refer to these other two functions as the init and run functions. Again, you can write this function in a Python. The init function is more about preparing your data because as you know after the data preparation, the model training, the model testing, validation, you select a model and then you need to deploy it, but the data preparation part has to be there because of course any different typologies of data that you're going to use, any new data that you're going to use, then it needs to be prepared in the same way. This first function that I was referring to is about preparing your data and making sure that basically the input data that you're using is going through the same process. This data, then you need to use them in order to feed your machine learning model. Then there is a second function that is called more like the run function. That is about making sure that your model knows how to read the data, and how to be run in order to produce your results. The combination of these init and run functions actually is really the core part of the model deployment. Most of the time you can use different tools and technologies that helps you to pack these two functions into to a file that is called the pickle file. You have really to think about this file as a folder where you have captured this information that I was referring to earlier; how your data needs to be prepared, and then how your data is going to be read by the model, and so that the model can produce the predicted results that you want. We refer to this as a pickle file.

How long does this process typically take?

Video Transcript

The process, it depends a lot. A lot of the time I use the answer, "it depends", because in machine learning, many things depend on the typology of data that you have and also on the type of problem that you're trying to solve. Generally speaking, I would say that creating just the two functions is just a matter of a few minutes, maximum an hour, because it's just about the data scientist needs to understand how to create these two functions and then you just write them in Python. That part, I would say it's a pretty quick part. However, when then you deploy your model, it depends, of course, on what is the model that you are using. So if it is like I would say, more like a complex model, it can take longer. Again, generally speaking, can be anything from, I would say, 30 minutes up to sometimes also one hour. But again, it depends on how much data you're using and the typology of the model that you are deploying.

What are the steps involved in the deployment process?

Video Transcript

There are many different typologies of the steps that you need to follow. I usually like to summarize these three steps in the following way. The first step is about registering your model; the second one is about preparing your models to be deployed, and finally, there is the deployment itself. Registering the model — that is the first step — is really about making sure that there is a sort of a logical container for one or more files that then you are going to use for running your model. For example, if you have a model that is storing multiple files, you can register them as a single model in your machine learning workspace. This is the space that you're going to use as a sort of environment for collaboration and experimentation. After the registration, you can then download or deploy the registered model and receive all the files that were registered there. Then there is the preparation to deploy that — honestly, it's probably the easiest step because it's about specifying the different assets, the usage, and also the computer target that basically you are going to use. And finally, there is the deployment itself of the model to the computer target. The deployment itself to the computer target is the part that I was mentioning before that is about writing this function, consuming actually is the function that you wrote before so that your model is then deployed into this pickle file. Again, think about this as a folder where you are going to see that there is the model that you created and also all the preprocessing steps that you need to perform on top of your data in order to make sure that your data is ready for your model. These are the three main tools, the three main technologies and steps that you need to follow in order to deploy your model. Again, register your model, prepare to deploy, then deploy the model to your computer target.

How do you detect errors in deployment?

Video Transcript

Most of the time, when you as a data scientist write the init and the run function, you immediately will notice if there is something that is not working well for your scenario. Why? Because the init function, this is the function that loads the model into what we call the global object and this function is run only once. Basically, most of the time when your docker container is starting, so when you need to basically input your data. Immediately there, if there are some problems that part, that function is going to give you an error. Because again, it means basically that your data is not processed in the right way. You will get an error from any different technologies or tools that you're using, because it's a Python function and as a consequence, you will get an error immediately. Another point, another moment where you can get a sense that your deployment is not really working, is at the run function, at the input data moment. This function uses the model to predict a value, because we are in a machine learning scenario, based on the input data. Here, we use also a process that is called the serialization and deserialization. So this means that the inputs and the outputs to the run, typically use what we call a JSON for serialization-deserialization. At this moment also, you will get an error immediately if you didn't get the right data and the model basically is not running in a proper way. Again, this is another moment which you immediately will get an error message or you will see immediately that your model is not working, and as a consequence, the deployment part didn't work out. This is good, because again, as I mentioned at the beginning that the init and run functions are somehow the first two things that you need to think about when you are ready to deploy your model. As you can understand, you're going to understand immediately, very early staging in the process that your model or the preparation of your data are not working in a good way.

Back to Table of Contents

Ask the Expert: Ask the Expert: Miriam Vogel on Mitigating Algorithmic Harms After Model Development

There are ways to reduce algorithmic harms when you monitor and maintain the model after deployment. In this video, Ms. Vogel describes what needs to be monitored, maintained, and updated once a model is being used in production.

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

Miriam Vogel President and CEO, EqualAI

Miriam Vogel is the President and CEO of EqualAI, a nonprofit created to reduce unconscious bias in artificial intelligence (AI) and promote responsible AI governance. Ms. Vogel cohosts a podcast, "In AI We Trust," with the World Economic Forum and serves as Chair to the recently launched National AI Advisory Committee (NAIAC), mandated by Congress to advise the President and White House on AI policy.

Ms. Vogel teaches Technology Law and Policy at Georgetown University Law Center, where she serves as chair of the alumni board. She also serves as a senior advisor to the Center for Democracy and Technology (CDT). Previously, Ms. Vogel served in U.S. government leadership, including positions in the three branches of federal government.

Most recently, Ms. Vogel served as Associate Deputy Attorney General, where she advised the Attorney General and the Deputy Attorney General (DAG) on a broad range of legal, policy and operational issues. Under the direction of DAG Sally Yates, Ms. Vogel led the creation and development of the Implicit Bias Training for Federal Law Enforcement. She also spearheaded the department's intellectual property (IP) efforts to identify and dismantle IP theft domestically and internationally, and she worked with the DAG to manage the multibillion-dollar budgets of the department's divisions;

resolve high-level challenges; and represent the department in briefings for White House, congressional, and GAO staff on policy initiatives and oversight matters.

Ms. Vogel served in the White House in two administrations, most recently as the Acting Director of Justice and Regulatory Affairs. She led the President's Equal Pay Task Force to promote equality in the workplace. Ms. Vogel also advised White House leadership on initiatives ranging from women, LGBT, economic, regulatory and food safety policy to criminal justice matters.

Prior to serving in the Obama administration, Ms. Vogel was Associate General Counsel at Dana-Farber Cancer Institute and practiced entertainment/corporate transactional law at Sheppard Mullin in Los Angeles. She began her legal career as a federal clerk in Denver, Colorado, after graduating from Georgetown University Law Center. Ms. Vogel is also a third-generation alumna from the University of Michigan.

After a model is built, are there mechanisms in place to mitigate algorithmic harms?

Video Transcript

We all know that when you're talking about artificial intelligence, it's going to continue to iterate, it's going to continue to build new patterns and have new answers, and we will have to learn what those are.

You have to continually monitor what is happening with your AI system to ensure that it is staying true to your intentions and that it stays true to your values, the company that you're working for, the organization's values, and that it's safe.

So are there tools -- this is a growing field. There are algorithmic auditors who are brought in at each stage in the development; it's just really the best way to do it. You want to bring in an expert when you're building the AI system to give you guidance along the way, but you also want to ask that person and continually ask yourself, "Where again will we need another audit? How often?"

And that will depend on the AI system. How many patterns is it learning? How quickly is it learning? How pivotal or high stakes are the use for which it's being deployed? And then you want to have a system in place to document, what did I test and when because down the road, other people are likely to be the ones doing that testing, whether it's a different auditing team, whether it's someone who has acquired that AI system and is looking at it with fresh eyes and they'll want to understand what data sets were used, what gaps were there in the population for whom it was tested, what were the use cases for which it was built so we can understand what might be outside of those contexts for which they'll need additional testing.

So in addition to the algorithmic auditors we mentioned -- so one of our senior advisors, for instance, is Cathy O'Neil, who wrote "Weapons of Math Destruction." She has an algorithmic auditing company called Orca and there are several others that are beginning to emerge. There's also tools that people can use.

So different companies put out different tools. There's a wide array that are starting to emerge, but one tool that is starting to be more widely deployed is an algorithmic impact assessment. And those are a series of questions that a leadership team or an outside expert helps you come up with to interrogate your AI system and understand where are the risks and the vulnerabilities, where are the opportunities, and where do we need additional testing or oversight.

We have one, for example, as I mentioned, on our website. If you look on our home page, you'll see an algorithmic impact assessment that EqualAI offers. Ours is based specifically on the NIST AI risk management framework that was released in January, and it really mostly reflects the questions that they say are best practice to ask.

But others have put up their models, too. Microsoft, I think Google; several companies have AIEs, algorithmic impact assessments, that you can make use of as well as other best

practices, such as model cards or other ways to routinely systematically document your AI audits and testing.

Back to Table of Contents

Tool: Upload Jupyter Notebooks to GitHub Repository

GitHub is a platform used to host and store code for version control and collaboration. It is free to use as an online directory or storage space for your projects. You may want to store your work from the program. This tool will guide you on how to download assignment files and then upload them to your own private code repository.

Download the Tool

Use this **Upload Jupyter Notebooks to GitHub Repository Tool** to help you with this process.

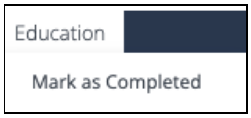
Back to Table of Contents

Assignment: Unit 5 Assignment - Model Selection for KNN

In this assignment, you will practice the evaluation phase of the machine learning life cycle. You will perform model selection to identify an optimal KNN classifier. You will evaluate the model's predictions using a confusion matrix and analyze the precision-recall curve which summarizes the precision-recall tradeoff for different classification thresholds.

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.

Please complete this activity in the course.

Back to Table of Contents

Assignment: Unit 5 Assignment - Written Submission

In this part of the assignment, you will answer five questions about selecting the correct models and approaches to solve your machine learning problems.

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 and exercises as well as the coding portion of the unit assignment.

Completion of this assignment is a course requirement.

Instructions:

1. Download the **Unit 5 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 5 Assignment document for review and credit.
5. Click the **Start Assignment** button on this page, attach your completed Unit 5 Assignment document, then click **Submit Assignment** to send it to your facilitator for evaluation and credit.

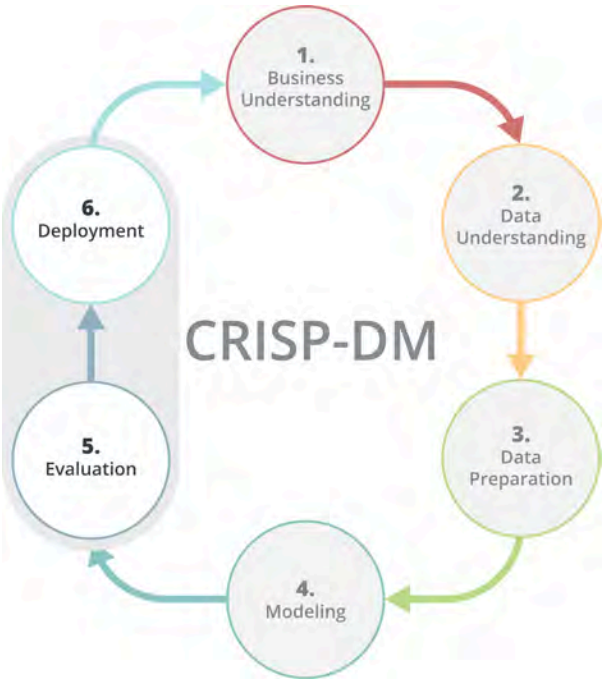
Back to Table of Contents

Module Wrap-up: Deploy Your Model

In this module, you explored various ways of deploying your model for production as well as things to monitor once your model is available to stakeholders. Remember to use the tool as a reference as you continue building future ML models!

Back to Table of Contents

Live Labs: Lab 5 Overview: ML Life Cycle: Evaluation and Deployment



In this lab, you will continue practicing the fifth step of the machine learning life cycle, where you perform model selection to identify an optimal logistic regression model. You will use multiple evaluation metrics to analyze the performance of models with different hyperparameter values. Using the built-in method from scikit-learn, you will perform feature selection on a logistic regression model with optimal hyperparameter values and analyze the results. You will then save your best-performing model using `pickle` and add it to your GitHub repository. You will be working in a Jupyter Notebook.

This three-hour lab session will include:

- **20 minutes** - Icebreaker
- **20 minutes** - Week 5 Overview and Q&A
- **20 minutes** - Breakout Groups: Big-Picture Questions
- **10 minutes -** Class Discussion
- **10 minutes - Break**
- **35 minutes** - Breakout Groups: Lab Assignment Working Session 1
- **10 minutes** - Working Session 1 Debrief
- **35 minutes** - Breakout Groups: Lab Assignment Working Session 2
- **10 minutes** - Working Session 2 Debrief
- **10 minutes -** Concluding Remarks and Survey

In Lab 5, you will:

- Build your DataFrame and define your ML problem.
- Create labeled examples from the data set and split the data into training and test data sets.
- Train, test, and evaluate a logistic regression model using scikit-learn's default hyperparameter value for C.
- Find the optimal logistic regression model using GridSearchCV.

- Train, test, and evaluate the optimal logistic regression model.
- Plot the precision-recall curve and the ROC then compute the AUC for both models.
- Practice the SelectKBest feature selection method.
- Save your best-performing model to a PKL file then add the model and data set to your GitHub repository.

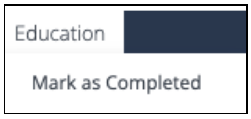
Back to Table of Contents

Assignment: Lab 5 Assignment

In this lab, you will continue working with the Airbnb NYC "listings" data set.

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.

Please complete this activity in the course.