Experimental Design for Data Analysis

by Janani Ravi

Start Course

Bookmark Add to Channel Download Course

Course Overview

Course Overview

(Music) Hi, my name is Janani Ravi, and welcome to this course on Experimental Design for Data Analysis. A little about myself; I have a master's degree in electrical engineering from Stanford and have worked at companies such as Microsoft, Google and Flipkart. At Google, I was one of the first engineers working on real-time collaborative editing in Google Docs, and I hold four patents for its underlying technologies. I currently work on my own startup, Loonycorn, a studio for high-quality video content. Providing crisp, clear, actionable points of view for senior executives is becoming an increasingly important role of data scientists and data professionals these days. Now a point of view must represent a hypothesis, ideally backed by data. In this course, you will gain the ability to construct such hypotheses from data and use rigorous frameworks to test whether they hold true. First, you will learn how inferential statistics and hypothesis testing form the basis of data modeling and machine learning. Next, you will discover how the process of building machine learning models is akin to that of designing an experiment. You will then see how training and validation techniques help rigorously evaluate the results of such experiments. Finally, you'll round out the course by studying various forms of cross-validation including both singular and iterative techniques to cope with independent identically distributed data and group data. You will also learn how you can refine your models using these

techniques with hyper parameter tuning. When you're finished with this course, you will have the skills and knowledge to build and evaluate models, specifically including machine-learning models using rigorous cross-validation frameworks and hyper parameter tuning.

Designing an Experiment for Data Analysis

Module Overview

Hi, and welcome to this course on Experimental Design for Data Analysis. In this module, we'll talk about how we can go about designing an experiment to analyze our data. We'll start off by understanding the journey that takes us from statistics to machine learning. We'll discuss here how it's a straight-line path that comes a full circle. We'll then talk about the object and process of hypothesis testing to evaluate proposed explanations for a phenomenon. We'll then discuss in some detail the two most popular statistical tests. We'll understand the T-test with tests for differences between two categories. We'll discuss the occurrence of type-1 and type-2 errors when we use statistical tests for hypothesis testing, and we'll then talk about using the ANOVA to test differences across multiple groups of data. From hypothesis testing we'll move on to machine learning models, and we'll talk about the four basic types of machine learning algorithms and how we would choose an algorithm based on our prediction target. And we'll round this module off by discussing and understanding the basic steps involved in building an ML model.

Prerequisites and Course Outline

Before you get started, let's briefly discuss what prereqs we need to have to make the most of your learning. This course assumes that you have some basic knowledge of Python programming. Some of the demos of this course will use Python. This course also assumes that you have some basic understanding of machine learning models. This is not a strict prereq though. And finally, the only map that you need to tackle the contents of this course is high school math. Let me give you a quick overview of what this course will cover. We'll first talk about positing and testing hypothesis. We'll talk about the process of hypothesis testing, the T-test, and the ANOVA test. We'll then get into hands-on mode and see how we can use the Azure Cloud platform to frame experiments and build models. We'll then talk about biases that might creep into our data model

and how we can account for these biases. This is where we'll discuss overfitted models and the bias variance tradeoff. We'll then discuss a number of different techniques to validate our model. The k-fold validation, stratified k-fold, repeated k-fold, will cover all of these. And finally, we'll close this course out by seeing how we can refine our models using hyper parameter tuning.

Connecting the Dots with Data

A powerful person once said, my mind is made up; don't confuse me with the facts. I don't think I've ever heard these words in my professional life. If you have, do let me know; I'd be curious. As someone who works with data, I'm often asked for my thoughtful fact-based point of view. Fact-based because it has to be based on real data that has been painstakingly collected. In order that my opinion be useful, it has to be thoughtful; it has to be balanced, and it has to weigh the pros and cons of the various options, and finally it has to be a point of view, a prediction, a recommendation, a call to action. These are the basic steps involved in any good data-driven business decision. A data professional typically has two sets of statistical tools available at her disposal. Descriptive statistics, which help her identify important elements in a dataset. Descriptive statistics are great for summarizing and quantifying the data that you have to work with. Using descriptive statistics on your sample data, you're able to infer information about the population as a whole. That's when we move into the realm of inferential statistics, which allow us to explain the important elements by our relationships with other elements, and these correspond exactly to the two hats that a data professional might wear. The first is find the dots, identify important elements in a dataset, and the next is to connect those dots that we've found, explain those elements by our relationships with other elements. This course on experimental design for data analysis will focus more on connecting the dots, and this can be divided into three components. The first is where you explore and pre-process the data that you have. Once you have a basic level of understanding of your data, you'll then posit hypotheses based on your data and build models using this data. And the sample of data that you're working with gives you information about the bigger picture. We'll then use this data to link to real-world data and scenarios and draw insights about the real world using this data. All three steps in connecting the dots are covered by related courses on Pluralsight. Exploring and pre-processing data is covered by these courses: Representing, Processing and Preparing Data; Summarizing Data and Deducing Probabilities; and Combining and Shaping Data. All of these focus on data cleaning, data pre-processing, and exploration techniques. These courses cover the first steps that you'll follow in understanding the relationships that exist in your data. The next step is to posit hypotheses based on what you've understood of your data and build models, and that's the objective of this course.

And finally, the new extracted insights from your data you want to link to real-world data and scenarios. That's covered in the course, Communicating Data Insights.

Hypothesis Testing

The first step towards data modeling is hypothesis testing. Hypothesis testing is that you posit hypotheses and then use statistical tools to determine that either your hypothesis was correct or wrong. We've spoken of the two sets of statistical tools used by a data professional, descriptive statistics and inferential statistics. How do we go from statistics to ML? There exists a straight line here that comes a full circle. Confused? Let's discuss this. Descriptive statistics are what you'll use to explore the data that you have available. You haven't developed any points of view, you just want to know what is out there. We'll perform uni-variant, bi-variant, and multi-variant analysis. We'll try and summarize and quantify the data that you have. From descriptive statistics, you'll move on inferential statistics. This is the step where you're trying to develop a point of view based on your data. You'll frame various hypotheses and test them out using statistical tools. You're not sure of your ground yet; you're tentatively evaluating many different points of view. Then you might move on to rule-based learning models. This typically involves a bunch of experts who've explored and understood the data and they've framed rules based on the data. This sounds great, but because it's performed by experts, some risks do exist. There's a risk of too much certainty; different experts will have their own pet theories. If you want to make sure that your model is objective and purely data driven, that's when you'll turn to machine learning models, and build models that change as the data changes, and when you're working with machine learning models, really you don't have your own point of view. You look to see what your model tells us. This brings us back full circle back to no points of view. The right place to start for us here is the hypothesis. A hypothesis is nothing by a proposed explanation for a phenomenon, something that you've observed in the real world, and you have some kind of explanation for why it occurs. The thing about the hypothesis though, is that it has to be objectively testable. Statistical tests exist to determine whether your proposed explanation is right or wrong. The singular for this is hypothesis and the plural form is hypotheses with an e. I have this in here because I tend to get confused and it's important that I say it clearly up front, hypothesis, hypotheses. So if you have a proposed explanation for a phenomenon that is a hypothesis, it needs to be tested using hypothesis testing. What does that involve? You'll first set up a null hypothesis represented by H0. This is what is considered true unless proven false. Let's say you're trying to test whether there exists a relationship between two variables, X and Y. The null hypothesis will basically say there is no relationship. The alternative hypothesis is what you're

trying to prove. This is a negation of the null hypothesis and you usually use this to assert a specific relationship. Yes, indeed, X and Y are related in some way. Once you have your null and alternative hypotheses set up, you'll then need to select a test that will allow you to evaluate whether the alternative hypothesis should be accepted or rejected. There are a vast library of tests available for this. You'll need to know which one to choose. Applying this test will give you a score that is the test statistic, and you'll convert this test statistic to a p-value. The p-value here is a measure of how likely it is that your test statistic was purely due to luck. In order to evaluate this you'll need to set up a significance level, which is typically 1% or 5%. If the p-value is under the significance level, you'll say that this test statistic was not due to luck. There does indeed exist a relationship between X and Y. And finally, you'll use this p-value to accept or reject the null hypothesis. If you have a small value, reject the null hypothesis and accept the alternative hypothesis. Remember that the null hypothesis says there is no relationship. When you reject the null hypothesis, you're saying there is indeed a relationship. Don't worry if this seems a little abstract. Let's try and understand this using a famous example, the lady tasting tea example. Was the tea added before or after the milk? Muriel Bristol, a famous lady back in the day, claimed that she could tell the difference, just by tasting every cup of tea. Could she or couldn't she? Let's apply hypothesis testing to find out. We'll first set up our null hypothesis H0, which claims that the lady cannot tell if milk was poured before the tea. Our alternative hypothesis is H1, the lady can indeed tell if milk was poured first. When we perform hypothesis testing, it's good practice to assume that the null hypothesis is correct unless proven otherwise. So we start off with the assumption that the lady cannot tell if the milk was poured first. Let's go back to the slide, which laid out the steps involved in hypothesis testing. The null hypothesis says the lady cannot tell the difference. The alternative explanation or alternative hypothesis basically says the lady can indeed tell the difference, can discern if milk was poured first. The next step is to select the test that will allow us to objectively evaluate our alternative hypothesis. The test that was actually conducted involved eight cups of tea, four of each type, and the lady go all eight correct. If you have eight cups of tea and four of each type, the total number of combinations that exist is 8 C 4 = 70 combinations. The test statistic converted to a p-value is the probability of getting even one of these right, which is 1.4 %. The significance level that we choose is part of the experiment design. Let's say it's 5%, 1.4 % is less than 5% so we can reject the null hypothesis. Based on our statistical test, we accept the alternative hypothesis, the lady can indeed discern if milk was poured before the tea. This famous experiment was conducted by Sir Ronald Fisher who is considered to be a founder of modern statistics. If you're using hypothesis testing, you ought to be aware of the errors that might occur. Along with the rules are situations where the null hypothesis is actually true or false. Along the columns we have the decision about the null hypothesis based on our

testing, whether we accepted the null hypothesis or whether we rejected it. It's possible that the null hypothesis was actually true and we ended up rejecting it. In our previous experiment the null hypothesis claimed that the lady could not tell the difference between tea and milk being poured first. A type-1 error occurs when we end up rejecting the null hypothesis based on spurious test results which are statistically significant. On the other hand, a type-2 error might occur when we fail to realize that the test for the alternative hypothesis was actually statistically significant and we should have rejected the null hypothesis, but we did not.

T-tests

A commonly used test for statistical significance is the T-test. We've spoken about the steps involved in hypothesis testing, and we've spoken of the fact that there is a vast library of tests available and knowing which one to choose is the difficult part. Statistics is a vast field of research and you should know that there are tests for pretty much everything, and these tests have been explicitly developed by statisticians to be sound. Now knowing which of tests is the right one to use for your use case is hard. Actually using these tests using libraries and other tools is very easy. When you talk of statistical testing, the first one that you're likely to encounter is the one that is the most common and simple one, the T-test. The T-test is typically used to compare data that belongs to two categories and used to learn about averages across these two categories. It will also tell us whether the differences that exist between these two categories are statistically significant. Let's see an example of where you would choose to use the T-test. Let's say you're comparing male babies and female babies based on their birth weight. Are they the same, or are male babies heavier than female babies at birth? Here we have two categories: male babies and female babies, and you want to compare their birth weights. Is the difference statistically significant? The T-test is what you'd use to find out. When you apply the T-test on the birth weights of male babies and female babies, you'll get a result that is the T-statistic. This is a score which indicates the difference in the means of the two populations. In addition to the T-statistic, the T-test will also give you a p-value, which will tell you whether the T-statistic that you got is significant or not. Your p-values under the significance threshold of 5% mean that the result that you obtained is not due to chance. A low p-value basically means that you can accept the alternative hypothesis, the difference in birth weight between male babies and female babies is significant. All statistical tests come with their own assumptions. The assumptions underlying the T-test are that the populations from which the samples are drawn are normally distributed, the samples are represented to samples of the population, and the samples have been randomly drawn from the population. The T-test cannot be applied if these assumptions don't hold true for

the samples of data that you're working with. Now T-tests work really well, and they're simple to use for two group comparisons: male babies, female babies, people who live in the U.S., people who live in Europe. All of these are examples of two groups. When you try to compare multiple groups, T-tests get tricky and they don't really work well. When you have multiple groups in your data, you need to perform several different T-tests for pairwise comparisons and this increases the likelihood of a type-1 error. The type-1 error is when you reject the null hypothesis based on spurious test results that are not statistically significant. If you have more than two groups, you'll perform hypothesis testing using ANOVA.

ANOVA

We've understood how T-tests work; in this clip we'll discuss ANOVA. Now T-tests are very useful to compare the differences between two groups or two categories. When you run multiple significance tests to compare across many groups, using T-tests is risky and can lead to type-1 errors. This is where you choose to use ANOVA or Analysis of Variance. ANOVA is what you'd use when you have multiple groups of data and you want to compare the differences across groups. The ANOVA test looks across multiple groups of populations, compares their averages or means to produce exactly one score and one significance value. The key difference between the T-test and ANOVA is the fact that ANOVA looks across multiple groups of populations. Let's understand this using an example. Let's say you're assessing the risk of diabetes amongst underweight patients, normal-weight patients, and overweight patients. A population as a whole is divided into three categories, not two; underweight, normal weight, and overweight. Remember that the T-test is used for comparison between two groups. If you want to use the T-test across these three groups, we'll need to perform multiple T-tests, which can lead to type-1 errors. Instead, if you use ANOVA as a statistical test, we'll just perform a single ANOVA test to know whether the risk of diabetes is significantly different between these groups. If you were to perform hypothesis testing, you first have to set up the null hypothesis. The null hypothesis basically claims that all groups of patients are at an equal risk of contracting diabetes. Our alternative hypothesis claims that all groups of patients are not at an equal risk of contracting diabetes. When we run the ANOVA test on our data, the output of ANOVA will give us an F statistic. The F statistic is what is used to compare the variance that exist between groups divided by the variance within a group. If the groups or categories in our data are very similar, F will be very close to 1. The variance between groups will be very close to the variance within the group. If on the other hand the groups are different, the value of F will be large, much larger than 1. In addition to the F-statistic, the ANOVA test also gives us a p-value for the result which tells us the significance of the F-

statistic. Smaller p-values indicate that the results are not due to chance and when you have a large F-statistic and small p-value, this means that the null hypothesis can be rejected and your alternative hypothesis can be accepted. Then you perform the ANOVA test on your data if you get a large value for the F-statistic and a small p-value under the five-percent significance level, you'll accept the alternative hypothesis and reject the null hypothesis. This basically means that all groups of patients are not at an equal risk of diabetes. Some groups are more prone to diabetes than others. If on the other hand, if you get a small F-statistic and a large p-value, greater than the 5% significance level, you'll accept the null hypothesis and reject the alternative hypothesis. The ANOVA test comes in two flavors. One-way ANOVA helps compare the averages or mean across two or more groups. This is where a single categorical variable is used to split the population into these groups. In our earlier example, we used just the weight of patients to split patients into three groups. When you use one-way ANOVA for hypothesis testing, once again there are a number of assumptions made about your sample and the population. The assumptions are populations are normally disputed, samples are representative, randomly drawn, and the variances of the population that we are working with are constant. Another flavor of the ANOVA test is the two-way ANOVA. This examines the influence of two independent variables on one continuous dependent variable. Let's say you have employees in your organization and you're taking into account both their age, as well as gender. Both of these are independent variables. Employees greater than and less than the age of 40, employees who are males and females. So you're actually working with four different groups, not two, and each group is based on the value of these two variables. This is when you choose to use the two-way ANOVA for hypothesis testing rather than one-way ANOVA. The two-way ANOVA test also tests for interactions within these two variables, age and gender.

Designing a Machine Learning Experiment

We spoke earlier of the path that takes us a full circle from statistics to machine learning. In this clip, we'll talk about common machine learning workflows and understand what exactly machine learning is all about. A machine learning algorithm is an algorithm that is able to learn from data. This algorithm has no upfront point of view, no hypothesis that is being tested. It simply looks at the data and makes predictions. The input to your machine learning model is a huge maze of data. This is the data that is used to train your model. The model looks at this data and has the ability to find patterns within this data, and these patterns help the model make intelligent decisions. For example, you could have a machine learning model that takes a look at all emails that arrive at a particular server. If you examine the contents of those emails and determine

whether the emails are spam or ham, and the action that you would recommend is move the email to trash if it's spam, and to the inbox if it's ham. The world of machine learning and data science is huge and is constantly expanding. Machine learning is being harnessed to solve a whole variety of problems, but at the very basic level there are four broad categories of problems that machine learning algorithms are called upon to solve. The first is classification; is this email spam or ham? Will stock prices go up or down? Does an individual have diabetes or not? Is this an image of a cat, dog, or a bird? The second category of problems where you might choose to use machine learning algorithms is regression. This is where you'll use ML models to predict a continuous value. What is the price of a house in this neighborhood given its features and location? What is the mileage of this car given its weight and horsepower? These are examples of regression problems. Machine learning models are also used to cluster your data. Let's say you have a vast expanse of documents and you want to know which of these relate to news articles, which of these are sports, which of these are technical articles and so on? That is an example of a clustering problem. Clustering involves finding logical groups in your data. And finally, machine learning can also be used for dimensionality reduction. Data in the real world typically has a huge number of features, not all these features might be significant or useful. Dimensionality reduction is the process of reducing the number of features that you can extract the most significant information from your data. If you are designing a machine learning experiment and building a model, this is the basic machine learning workflow that you'll follow. Yes, there are a lot of steps in here, but you'll find that they flow logically from one to the other, and we'll examine each step in detail. When you design your modeling experiment, the first thing that you need to do is look at what data you have to work with. What is the raw data like, and what is it that you're trying to predict using this data? Once you know this, you can move on to the next step where you prepare your data for modeling. You might need data from different sources. You'll need to ensure that all of your data is available in one place to train your model. The next step is data preprocessing where you clean the data, fill in missing values, remove outliers, etc. If you have text or image data, you might need to convert them to numeric form for analysis, and you might standardize and scale your data so that the models you build are more robust. Once this is done, you'll then choose the algorithm that you want to fit on your data. The algorithm will defer based on whether you want to perform classification, regression, clustering, or dimensionality reduction. It'll depend on the kind of data that you're working with and the characteristics that you want in your model. Once you've chosen your algorithm, you'll then fit a model by training your algorithm on training data. The process of training is to find the model parameters. Once the training process is complete, you'll move on to validating your model, evaluating whether it's a good one. There are of course several validation techniques available. You'll pick one that makes sense and choose a

metric to evaluate your model, and you will examine, fit, and update the model if needed. Based on the metric that you've chosen to evaluate your model, you'll see whether this model is satisfactory. If no, you'll update the model and follow this entire process once again. Iterate until your model has been finalized. This rinse and repeat process where you train your model, tweak its parameters, validate your model, evaluate it, and then check once again as a part of your experiment design, and hopefully there will be a point where you're happy with the model that you have. If you're satisfied, you'll use this model for predictions, and as new data comes in, this new data will be fed in to improve your model, and this completes the basic machine learning workflow and the design of our experiment.

Summary

And with this we come to the very end of this module where we spoke about how we would design an experiment for data analysis. We spoke of descriptive statistics to explore our data and inferential statistics to understand our data. We studied hypothesis testing to evaluate proposed exclamations for a phenomenon. We studied the steps involved in hypothesis testing, and we understood how the T-test works to test for differences between two categories. We spoke about type-1 and type-2 errors in hypothesis testing and how we could overcome type-1 errors when we have multiple groups of data. When data is divided into more than two categories, rather than use the T-test, you'll use the ANOVA to test for differences across multiple groups. Once you've tested your hypothesis, you'll use these hypotheses in framing rule-based algorithms, but even better is when you have no upfront point of view about your data and you build machine learning models, which are driven by data. We discussed the four basic types of machine learning algorithms. There are of course several more, and we spoke about how we would choose an algorithm based on our prediction target, whether it's a classification model or a regression model. And finally, we discussed in a lot of detail the basic machine learning workflow and clearly understood the steps involved in building a model. In the next module, we'll put what we learned to practice. We'll see how we can use Azure's machine learning studio to fit a regression model for house price prediction.

Building and Training a Machine Learning Model

Module Overview

Hi, and welcome to this module on Building and Training a Machine Learning Model. In this module, we'll apply some of the techniques that we studied in the previous module for experimental design of your data. We'll work on the Azure Machine Learning Studio, which is a very intuitive, drag-and-drop user interface to build ML models. We'll write absolutely no code, but we'll still be able to collect and process the data that we need to train our model. We'll define features from our data and use these features for prediction. We'll then design an experiment for predictive analysis. In this case we'll build a regression model. We'll choose the algorithm that we want to train, and then we'll train, score, and test our model. You'll be quite surprised at how much we're able to achieve using absolutely no code. This is because of the Azure ML Studio. This is browser-based visual drag-and-drop no-code environment for building ML models. This democratizes ML to such an extent that it's accessible to all. The general trend in data nowadays it the democratization of machine learning. Cloud platforms offer a suite of tools for all general enterprise needs. On the other hand, we have data platforms, which offer a suite of ML, AI, and Big Data tools for data professionals. The Azure ML Studio serves to bring both of these together in one environment, giving your entire enterprise access to machine learning and AI modeling environments, allowing you to build models without writing any code.

Getting Started with Azure ML Studio

In this demo, we'll get started with designing our machine learning experiment with Azure ML Studio. In order to perform the demos of this course, you need to have an account on the Azure Cloud platform. Go to portal.azure .com. I'm already logged in with my Cloud user account. You need to make sure that is billing enabled for your Azure account, though most of the resources that you'll consume as a part of this course should cost you under a dollar or two. Here is the main Azure services dashboard and on the left side we have a navigation pane, allowing you to access to different services on the Azure Cloud. Let's go ahead and click on the option to create a new resource. A resource on the Azure Cloud could be any. It would be a SQL database, a virtual machine, but what we're looking for is a Machine Learning Studio Workspace resource. Type this into the search bar and click through on the auto-complete option and this will take you to the page for the Machine Learning Studio Workspace. The Azure Machine Learning Studio is a cloud-based environment on which you can run your ML experiments. You can prepare data, train, test, deploy, manage, and track your machine learning models. All of your resources for Azure machine learning are contained within the top level workspace resource. This is a centralized place to work with all of the artifacts that you create when using the ML service. Click on the Create button here

and you'll be taken to a simple form where you can create a new workspace. I'm going to call this LonnyWorkspace01. I have a Pay-As-You-Go subscription for my Azure account so that's the only choice I can make here, and I'm going to select a new resource group within which this workspace will reside. A resource group on Azure is a container that holds all of the resources for a particular solution, in this case our machine learning experiment. You can view the billing details for all of the resources in a resource group together, have a unified dashboard, and so on. I've named my resource group MLStudioRG. Click on OK, and let's move on and specify a location for our workspace. I'm going to go ahead and choose Japan East. There is an option here to create a new storage account associated with your workspace. I call this the loonyworkspace01storage. It's the standard here, I don't want anything fancy, and I'm going to create a new workspace service plan. The service plan determines how much you'll be charged for the resources that you consume. I haven't selected a pricing tier yet. Click through to web service plan price tier, and I'm going to go with the DEVTEST Standard option. What we are building in our demos is just a prototype so just two compute hours and a thousand transactions is enough for us, and it won't cost us any. Click on Select to make this choice, and you can go ahead and create this new workspace once your pricing plan tier has been selected. You might have to wait for around a minute until your workspace and your resource group has been deployed. You can then select the Go to resource option, and here is your LoonyWorkspace. All of the machine learning experiments that you conduct on Azure have to be within the workspace. The first option is the one that we'll choose here; we'll launch the Machine Learning Studio. Azure ML Studio is a visual browser-based drag-and-drop authoring environment to develop machine learning models. You need to write absolutely no code to build your solutions. This tool makes it very easy for you to run different experiments with your ML models and fine tune the parameters of your model. Click on the Launch Machine Learning Studio, and you'll be taken to the azureml.net page for your region. This is Japan East for me. Click on Sign In here. I've already signed in with my Azure account. This takes me directly to Azure Machine Learning Studio, and you can see that I'm working within the LoonyWorkspace01. We're on the Experiments tab here. An experiment in Azure ML is the main entry point into experiment with the Machine Learning Service. The experiment acts as a container of sorts for all of the trials that you conduct of your ML models. There are no sample experiments as well. Let's head over to Notebooks. These refer to Azure notebooks on the cloud. Azure notebooks can be thought of as Jupyter Notebooks hosted on the Azure Cloud, which come with all of the data science libraries that you need pre-installed. Jupyter Notebooks offer an interactive programming environment for Python and our programs. Let's head over to the Datasets option. Here is where you'll upload all of the data that you'll use to train and evaluate your machine learning models. Now if you want some sample datasets to play around with, if you

click on samples, you'll see a bunch of these are already available on the Azure Cloud. We can go with any of these datasets to build your prototypes. Once you have a fully trained machine learning model that you can then use for prediction, you can click on the Trained Models tab and find the model listed here.

Loading and Visualizing Data

In this demo, we'll load and explore the dataset that we're going to be working with to build and train our machine learning model. Here we are on the Datasets page of our Azure Machine Learning Studio experiment, and at the bottom here we have the option to create a new dataset. Click on New, and I'm going to create a new dataset from a local file. This file is available on my machine. This dialogue allows you to choose a file from your local machine to upload to the Azure Cloud. The data that we're going to be working with is the King County housing prices dataset. The link below shows where this dataset is available on kaggle.com. This is basically a dataset which contains the various features of houses located in King County in Washington State and the corresponding prices for those houses. We'll build and train a regression model that allows us to predict the price of a house based on its features. The file is a CSV file where the first row is the column header, Generic CSV File with a header if the format of this dataset. Let's specify a description here, House Sales in King County, USA, and click the checkmark. This dataset has been successfully uploaded on the Azure Cloud and is now available for use within our experiment. Let's head over to the Experiments page and click on the New option here to create a new machine learning experiment. We'll choose the Blank Experiment template. This will take you to the actual ML experiment, which as you can see is very visual. Let's first name our experiment KCHousePricePrediction. As you can see, it's an entirely drag-and-drop UI where the machine learning workflow is present in the form of a graph with nodes and directed edges. Off to the left are all of the modules that can be used to perform specific operations. Merely drag these modules onto the main pane to build up our machine learning workflow. Any node in the machine learning workflow is associated with certain properties and the node then selected the properties show up in the Properties family off to the right. I'm going to close this for now so that we have more room to work with. I'll go ahead and close the panel for the experiment items or the processes as well. The Mini Map here at the bottom left is a way for you to zoom in on different portions of your machine learning workflow. I'm going to minimize this as well since we don't really need it. Now let's get started with the dataset that we've loaded in. Click on the Saved Datasets and under that, if you take a look at the My Datasets option, you'll see that the King County housing data that we loaded in is available. Select this data file and drag it onto the center

pane, and simply drop it and this sets up the first node in your directed directed basic cuff that represents your ML workflow. In order to export this dataset you can simply select the node by clicking on it, right-click, choose the dataset option, and choose the Visualize option in the sub menu. This will bring up a dialogue, which will give you basic information about all of the features in your dataset, allow you to use statistics and visualizations. You can see there are about 21, 000 rules and 21 columns in this data. You can see that there is a column giving us the price of this house in U.S. dollars, the number of bedrooms, bathrooms, the living square feet, the square feet basement, number of floors, waterfront, whether it has a view or not, and so on. There's a whole bunch of information available on these houses and this is what we're going to use to train our ML model to predict the price of a house with these features. I'll scroll to the left and select the price column, and off to the right you'll see basic statistics about the price. You can see that the mean housing price is about $540, 000 and the median is about $450, 000. It's a numeric feature with no missing values, and if you scroll down you will see a histogram of the distribution of prices in this dataset. If you're interested in any of the other columns in this dataset, simply select that column, let's say square feet living, and view basic statistics for this column and the visualization, the frequency distribution histogram. Most of the houses in this dataset have a living square footage of between 1600 and three-thousand square feet. If the data in certain columns is better visualized as a box plot, simply click the box plot icon and the representation of the data will change. Let's now select the price column and see how the distribution looks in a box plot. The prices of most of the houses in this dataset are under 1.5 million, but there are a significant number of outliers as well. Let's explore a few more columns before we are done. I'm going to select the bedrooms column. You can see that the median number of bedrooms is 3, the average is around the same, and you can see that there are a few houses which are very, very large, with up to 35 bedrooms. If you take a look at number of bathrooms in the various houses you'll see a similar distribution, and most houses have a little over two bathrooms, but if you scroll down and look at the box plot, you'll see that there are a number of houses which have five or even eight bathrooms.

Exploring Relationships in Data

Now that we've understood and explored the data that we are working with, let's now continue with building our machine learning workflow. Select Data Format Conversions and under that we have the option to convert to dataset. All of the data that you feed into your ML model has to be in the form of a dataset, and there is a built-in process to perform this conversion. Select the Convert to Dataset option and drag it onto the center pane and use your mouse to connect the

csv file to the Convert to Dataset node. And this is how you set up the first two nodes of your machine learning workflow and connected them by a directed edge. You can execute the processes that you've set up. Simply go to the Run option at the bottom, and you can run all of the workflow or a selected portion of the workflow. I'm going to choose the Run option to execute the entire workflow, and the green checkmark shows me that this process is now complete. Now that we have the dataset, under Statistical Functions on the left navigation pane, it's possible for us to summarize the data that we are working with. Select the Summarize Data option and drag it onto the center pane, and connect the previous node to this particular node. Once we've fed in the data that we want to summarize, select the Summarize Data node, and go to the Run option at the bottom and choose Run selected. This will simply execute the process to summarize your data, and once you have your summary available, you can always visualize it by right-clicking on the summarize data node and going to Result dataset Visualize. The features in your dataset make up the rule here and you'll get some of these statistics for each feature, such as the count, the unique value count whether there are any missing values or not, min, max, mean, median, and so on. The summarize data process is a useful function allowing you to get a quick sense of your data. You can close out this dialog and let's go back tour machine learning workflow under Data Transformation. I'm now going to manipulate this data in a specific way. I want to work with just a few selected features in our dataset, which is why I'll choose the Select Columns in Dataset, drag it onto the center screen, and connect it to the previous mode. All of the features in our input dataset may not be relevant. Select columns in dataset will allow us to pick and choose features. The little warning icon here tells us that we need to select the features or columns that we're interested in, for which I'm going to bring up the Properties panel. After selecting that node, I'll bring up the Properties panel and launch the column selector. All of the columns in the dataset are present on the left. Select the column, use the arrow key to move this over as a selected column that you're interested in. Following exactly the same process, I'm going to select a number of different columns. Go ahead and select columns from the left, use the arrow key, and move all of the columns that you're interested in over to the right. Here are my selected columns. You don't need to have exactly the same columns to train your model, but this is a good subset. With the columns that we're interested in selected, we can now close the Properties panel and add the next step in our machine learning workflow. Head over to Statistical Functions once again, and there is an option there to compute linear correlation. This will allow us to further explore our dataset. In order to view the linear correlations we'll need to select this node and then choose the Run selected option in order to execute all of these nodes. Once execution is complete, you can right-click on a particular node and visualize the results of that node, and this will bring up the dialog that we are familiar with. You can see that every feature is present in a

row, as well as a column, giving us a correlation matrix, which tells us how every pair of features is related. The correlation tells you whether there exists a linear relationship between a pair of features, and you can select a particular column, and you can visualize its relationship with any other column by choosing the compare to option. Compare the price to square foot living, and you'll see that the result is a scatter plot. The structure of the scatter plot makes it very clear that there does exist a linear relationship between the living space and the price of a house. Let's perform one more comparison to understand this price versus number of bedrooms, and you can see that this scatter plot also displays a linear relationship.

Preprocessing and Preparing Data

In this demo, we'll see how you can preprocess your data before you feed it into train your machine learning model. Here is what our computation graph or machine learning workflow looks like now. Under Statistical Functions is there is an Apply Math Operations node that you can use to perform math operations to process your data. Set this up as a part of your machine learning workflow, and let's take a look at the map operation property. I'm going to apply a map operation to compare the value in the Year Renovated column to convert it to a Boolean value, whether a house has been renovated or not, true/false values. The result of this map operation that I specified will be a Boolean value. The math operation that I want to perform is a comparison operation so I choose the category Compare and the comparison function that I apply is to check whether the Year Renovated field is greater than 0. Choose the GreaterThan option and then we specify the value that we want to compare to. I don't want to compare to a column, I want to compare a constant, and the constant to which I compare will be 0. So far I've only specified the map operation that I want to apply, but to which column? That's where we'll use the column selector. Bring up the column selector and let's switch over from column types to column names. This will give you a textbox where you can input the columns on which you want this map operation to apply. I'm only going to select the year renovated column here. Click on the checkbox to apply this. We want the conversion of this column using this map operation to be applied in place. The output mode should be Inplace, so that's what I pick in this drop-down. Having specified this math operation on the year renovated column, let's go ahead and Run selected, and this math operation will be applied to year renovated. You can take a look at the results by right-clicking and going to Visualize Results dataset. If you select the year renovated column in this dialog, you can see that it has two unique values. It has been converted to binary true/false values indicating whether a particular house has been renovated or not. Let's continue preprocessing our data. Head over to Data Transformation and under Manipulation, this time I'm

going to edit the metadata of some of the fields in our dataset. Choose the Edit Metadata option, drag it to the center pane, and connect it to the output of Apply Math Operation. Select the Edit Metadata node, and let's use the Properties pane to edit the metadata for certain columns. Launch the column selector. I'm going to select some of the categorical values in our dataset, discrete values such as whether a house is located at a waterfront or not, whether it has a view or not, and whether the house was renovated or not. These are three columns that I want to convert to categorical values. In the Categorical drop-down choose the Make categorical option and that'll automatically perform this conversion for you. With the Edit Metadata node selected, go ahead and choose the Run selected option to execute this portion of the workflow. Once the execution is complete, you can right-click and go to Visualize Results dataset, and you can select each of the columns that we just converted, and you can see that the feature type is now a categorical feature. This is for the waterfront column, zeroes and ones, these are all discrete categorical values. Now our year renovated column comprised of binary true/false values. This is a categorical feature. You can convert categorical features to indicate the values. Indicator values are a numeric representation of your categorical values using one hot notation. Select this newly added note in our workflow and bring up the Properties panel and launch the column selector so we can specify which column we want to convert to an indicator value. The year renovated column is the one that I'm going to select. This was originally a categorical feature with Boolean true/false values. I'm going to overwrite the categorical values to convert it to an indicator feature. Instead of having discrete categorical values, in this case Boolean categories, I'm going to represent it in one hot notation. Go ahead and run the selected node to convert the categorical column to an indicator value, and once it's done, right-click, go to Results dataset, and visualize and notice that year renovated is now two columns. The data is represented using one hot representation. Year renovated 0 will be 1. When the house has not been renovated, year renovated 1 will have the value 1 if the house has been renovated. Let's close this column and go back to preparing and cleaning our data. The next step is to take care of any missing values that might be present. Under data transformation and manipulation, there is an option to Clean Missing Data. I'm going to drag and drop this node onto my main machine learning workflow, and connect the output of convert indicator values to this node. Select this node and launch the column selector in the Properties panel. We want the Clean Missing Data operation to apply to all columns, except the default selection and let's specify the cleaning mode. So what do you want to do if there is missing data? I'm going to remove the entire row. This is by far the easiest way to tackle missing data. There are other interesting options here as well. You can replace the missing data with the mean median mode or some other customization, but Remove Entire Row is what we want to do here for this workflow. Go ahead and use the Run selected option to clean your

missing data. And once this node has finished execution, you can right-click on the node and go to Visualize, and see the results of your cleaned dataset. You can see that we have fewer rows now as compared with earlier. All of the rows with missing data have been removed.

Building and Training a Regression Model for Price Prediction

With all of our data cleaning and processing complete, in this demo we'll finally build and train our machine learning model. We're going to split our input dataset into training data that we'll use to train our model and test it and that we'll use to evaluate our model. Under Data Transformation, go to Sample and Split. Under this you'll find an option, which lets you split your data. The training data is what we'll use to train our model parameters and the test data is data the model has never seen before. That's what we'll use to measure how our model does. Drag the Split Data node onto our machine learning workflow, and connect the output of the Clean Missing Data to the input of our split data. The clean data is what we want to split. Bring up the Properties panel. The splitting mode that we're going to use is to Split Rows. So we are going to split horizontally. The fraction of the data that I want in my first output is 0.8. Eighty-percent of the data to train the model, 20% to test the model. With the Split Data node configured and selected, go to the Run option and choose Run selected so that our ML workflow will actually perform the data splitting. Notice that Split Data has two outputs. The first of these is Results dataset1. That is the 80% training data and the second is Results dataset2, 20% which forms the test data. Select the test data output, right-click, and choose Visualize, and you can visualize and you can visualize just the test dataset. You can see there are just about 4000 rules, 20% of the entire dataset. If you close this dialog and select the output node for the training dataset, right-click, and choose Visualize, and you'll be able to see that 80% of the records are present in the training data, 17, 289. We are now finally ready to build and train our machine learning model. Using the left navigation pane, go to Machine Learning, expand it and choose the Initialize Model option. Now the kind of model that we want to build is a regression model, which we'll use to predict a continuous numeric value; the price of house in King County. There are a number of different algorithms that Azure Machine Learning Studio supports to build a regression model. Of these, I'm going to go with the simplest. We'll perform linear regression on our input data. Select the Linear Regression model and drag it onto your ML workflow screen. Now that you've chosen the kind of ML model that you want to build, we need to specify the action that you want to perform using this model. We first want to train our model. Choose the Train option, and under that you'll find a node that allows you to train your ML model. Drag this Train Model option onto your machine learning workflow and one input to this model is your training data. Connect that output of the Split Data

node, which 80% of our data. This is the training data for our machine learning model. The next input to the Train Model process is a model that you want to train. This is the Linear Regression model. Connect that as an input to Train Model as well. What do we want to train this model to do? Predict housing prices. We need to specify this using the Properties panel, bring up Launch column selector, and specify that the price is what you want this model to predict. Move that over to the selected columns. And with this, I have the complete workflow needed to train my machine learning model. Go to Run and this time I want to run the entire workflow, not just a few selected nodes. You might have to wait for about a minute or so for the entire training process to be completed. Once you have the Train Model, you'll find a checkbox next to that node. Go to Train Model and Visualize, and this will allow you to see what this model looks like. We set a linear regression model with a bias term, and there was some regularization involved in order to ensure that the model did not over fit on the training data. All fitted models are those which perform well in training, but poorly in prediction. Regularization helps mitigate over-fitting. If you scroll down below, you'll find the weights that our linear regression model assigned to the various input features. Each of these weights represent how significant that particular input feature was in predicting the price. Now that we have a train model, let's evaluate how this model does on test data, data it hasn't seen before. Under machine learning, you'll find a link for Score. Here is where you can score and evaluate how your trained model has performed. Choose the Score Model node and drag it onto your main machine learning workflow, and connect the trained model to your score model process. We are going to evaluate our model on the test data. From the Split Data node, choose the output for the test dataset and connect to Score Model. Your model will be evaluated on data it has never encountered before. That's when you know whether your model is robust or not. Select the Score Model node, go to Run selected, and score your trained model, and once this is complete, you can right-click on the Score Model, go to Scored dataset, and let's visualize the results. The score model operated on the test data with about 4000 rows, and if you go to the very right, you'll see that a new column has been added. These are the Scored Labels. The scored labels that you see here are the predicted prices from our machine learning model for all of the houses in the test dataset. You can see that the mean predicted value is about 540, 000, and the median is around four-hundred-eighty-one-thousand. Let's compare the mean and median of the predicted housing prices with the actual price. This will give us an idea of far off our model is. We can see that the mean is 543, 000, very close to the predicted mean of 540, 000, and the median is 453, 000, close to the predicted median of 481, 000. So this model seems to be pretty good. This is just an estimate. We need a more objective evaluation of this model, and for that, we go back to machine learning, and under that the Evaluate option, and choose the Evaluate Model node and drag it onto your main ML workflow, and connect this node to the

output of the Score Model process. This will take the scored labels and see how good our model is using an objective measure. Run the evaluation of our model and once that is complete, right-click on the Evaluate Model node and let's visualize the results. An objective measure for how good your regression model is, the R-squared score, also call the coefficient of determination. The R-squared score measures how well the linear model that we've fit on the training data captures the variance in the underlying data. This is a number between 0 and 1 and an R-squared score of 0.64 means that this model is pretty decent. Now that we have a fully trained model, we can save this trained model and deploy it later if we want to. Go back to the Train Model, right-click on it, and choose the Train Model, Save as Train Model option. This will bring up a dialog where you can specify a name for this model, KCHousePricePrediction is my name, and once your model has been saved, you can head over to the main navigation pane. Click on Experiments. This will show you that the machine learning model that we just built and trained has been saved as an experiment, but if you switch over to the Trained Models link here, we'll find that this model is available as a trained model ready for deployment. In this course, we won't go into how we can deploy and use this model for prediction in the real world. We'll focus on the experimental design of this model.

Building and Training a Regression Model in Python

In this demo, we'll build and train a machine learning model in Python using the scikit-learn library. We'll build the same linear regression model that we built on Azure ML Studio. We'll write our code on Azure notebooks. We have no notebooks present at this point in time. We'll create a new notebook by clicking on the New link here at the bottom of the screen, and the notebook that we want is a Python 3 Blank Notebook. The code that we'll write will be using Python 3. Specify a name here, KCHousePricePrediction, and here is our Jupyter notebook hosted on the Azure Cloud. I want to make sure that I'm using the latest version of certain Python packages such as matplotlib so I get the version that I want using pip install. Just like you would with Jupyter notebooks on your local machine, hit Shift+Enter to execute the code in any cell. I'm going to pip install the scikit-learn library as well. The latest version at the time of this recording is 0.20 .2. In addition to matplotlib, I'll use the seaborn library for visualization. I'm going to pip install seaborn 0.8 .1. Once all of these Python packages are available within our notebook, let's set up the import statements for these. We'll use the seaborn and matplotlib visualization packages to visualize relationships in our data, and we'll use the scikit-learn linear regression estimator object to build a regression model. When you're working on Azure notebooks, you can access datasets from within your Azure workspace. From Azure ML import a workspace object and instantiate a new

workspace, and within this workspace we'll access the King County housing data. The workspace that we've instantiated is the current workspace that hosts this notebook, and this is the dataset that we work with on Azure ML Studio. I'm going to convert this dataset to a Pandas DataFrame using the to_dataframe method. Let's take a look at a few sample records in this data frame within this Jupyter notebook. Here is the dataset that we've built and trained our model on in Azure ML Studio, and we'll do the same thing in Python. The shape of the dataset shows us that there are about 21, 613 records and 21 columns. We feel like some of these columns are not useful, which is why I'm going to go ahead and drop them; id, date, zip code, lat, and long. We won't be using these features to train our machine learning model. Once the houseprice_df contains the columns that we're interested in working with, let's perform some data cleaning and preparation. If the value in the year renovated column is greater than 1, that means the house has been renovated. We'll set it equal to 1 so that the year renovated now has numeric labels to indicate categories. Zero if the house hasn't been renovated, one if it has. We also want to clean up any rules with missing data. This is very straightforward in the Pandas DataFrame. Simply call drop any and if there are any records in our data frame with missing field values, those will be dropped. Let's take a look at the data types of the different features that we're going to work with. You can see that all features are numeric, either integers or floats. Some of these features we know are categorical values such as waterfront, view, and year renovated. I'm going to iterate through these columns and convert them to categories. Access each of these columns in our data frame and use the Pandas astype function to convert these to categorical values. If you take a look at the data types now, you can see that waterfront, view, and year renovated are categories or discrete values. You can use Python code available in Azure in order to convert this data frame into a dataset. From Azure ML DataTypeIds and you can simply call ws.datasets .add_from_dataframe to convert what we have in a data frame into a dataset that can be used for Azure ML. The type of this dataset is DataTypeIds.GenericCSV. The name of this cleaned dataset is houseprice_data. Switch over to your Azure ML page and under the Datasets option, you'll find that house price data is now available for ML Studio to use. We won't really use it right now. We'll be needing it later on in this course. Let's switch back to your Azure notebook to continue writing Python code to build our regression model. We'll now visualize the relationships that exist in our data using matplotlib. Here is a simple matplotlib scatter plot showing the relationship between the number of bedrooms and bathrooms across all houses in our dataset. The shape of the scatter plot makes it very clear that as the number of bedrooms increases, bathrooms also tend to increase. Let's explore another pairwise relationship between the living square footage and the price of the house. Once again, this shape of the scatter plot makes it clear that this is a linear relationship. The larger the house, the higher the price. Every home in our dataset is assigned a

grade by some kind of inspector. Let's take a look at the relationship between the grade of the house and the price, and once again it's clear; the higher the grade, the higher the price. The best way to view pairwise relationships across all of your data is to use a correlation matrix. The correlation between two variables is set to be positive if an increase in one results in a corresponding increase in another. It's negative when if one increases the other tends to fall. The corr function on the Pandas DataFrame calculates the correlation matrix for all of the columns. You can see that the correlation of every column with itself is 1, therefore perfectly positively correlated. It should be no surprise to you that square feet living is positively correlated with the price of a house. What's kind of interesting here is that the number of floors in a house is negatively correlated with the size of the basement of the house. Houses with larger basements tend to have fewer floors in Washington State. A correlation matrix can be visualized using a heat map. Call sns.heatmap and pass in the correlation matrix that you had calculated before. The heat map is a colorful grid where the colors indicate positive or negative correlation. Positively correlated values tend to be in deep red, whereas negatively correlated values tend to be purple or black. Now that we've explored our data, let's move on to the actual process of building and training our model. The buy values or what we want to predict are the house price. Let's store the price value in the buy variable. All of the features that we'll use to train our model, I'll go ahead and store it in the X variable, everything other than price. We'll now split our data into training data and test data. We'll do this using the train_test_split function in scikit-learn. Just to change things around a little bit, I'm going to use 60% of the original data to train our ML model and 40% to test any value with it. So if you look at the shape of the training data, we have about 13, 000 records to train our model, and around 8, 500 records to test our model. Now building and training a model in scikit-learn is very straightforward. You simply instantiate a linear regression object. This is an estimator object and call fit on the training data. This will automatically start the training process. X_train contains the features to train our model and y_train contains the original labels. Once we have a fully trained model, we call regression.score and pass in the test data. This is what will allow us to evaluate our model on instances it hasn't seen before. The score function on our train model will spit out the R-squared score of the model on the test data. R-squared if you remember is what we can use to objectively evaluate this regression analysis. An R-squared score of .66 is very similar to what we got from our Azure ML Studio, and it's a pretty decent score.

Summary

And this demo gets us to the very end of this module on Building and Training a Machine Learning Model. In this module, we worked with the Azure ML Studio, which was a browser-based visual drag-and-drop, no-code environment for building ML models. ML Studio is simple, intuitive, and easy to use, even for those who can't write code. This serves to democratize machine learning making it accessible to all. We used Azure ML Studio to collect and process the data that we needed to train our machine learning model. Once we got all of the data, we defined features from this data and we designed an experiment for predictive analysis. We performed regression analysis on housing prices and used it for prediction. We saw how Azure ML Studio offers a variety of machine learning algorithms that we can work from. You could choose any of these algorithms; we went with simple linear regression. We then trained, scored, and tested our model. Most of our focus in this module was on Azure; however, at the very end we also built and trained a regression model using the Python scikit-learn library. In the next module we'll discuss classification models and the metrics that you can use to evaluate them. Specifically we'll discuss accuracy, precision, and recall, and the ROC curve.

Understanding and Overcoming Common Problems in Data Modeling

Module Overview

Hi, and welcome to this module on Understanding and Overcoming Common Problems in Data Modeling. We'll first start off by identifying common biases that might exist in your model, typically overfitting. We'll then talk about different techniques that you can use to mitigate these biases: regularization, cross-validation, ensemble learning, and dropout. We'll discuss in detail overfitted models and what they mean. We'll talk about what it means for a model to have high bias and high variance, which will take us into the next topic where we discuss the bias/variance tradeoff. We'll then turn our attention to evaluating classification models using metrics like accuracy, precision, and recall. If the dataset that you're working is a skewed one, you'll find that accuracy is not a great metric to evaluate your model. That's when you'll need to use precision and recall. We'll see how we can have our model make the right tradeoff between precision and

recall by using the ROC curve. And finally, we'll bring all of this together in a hands-on demo, which will use Azure ML Studio to build a classifier model.

Overfitting and Techniques to Mitigate Overfitting

Just because you're using ML doesn't mean you have a good model. Machine learning models can be overfitted on the training data, which means they perform poorly in the real world clearing prediction. Talk about what it means to have an overfitted model. Here is your challenge. Find the best curve that passes through these points. Now you can say that a curve has a good fit if the distances of points from the curve are small. On the face of it there's nothing really wrong with this statement. It seems like a good way to evaluate whether our curve fits our underlying data, but the fact is we could draw a pretty complex curve. You can take this idea that we have of the best fit curve to an extreme and you can have a really complex curve that passes through every single data point, but if this really complex curve represents your model and you plan to use this model for prediction, given a new set of points, this curve might perform quite poorly. The original points on which you fit this curve are our training data, and these new points that we've added in are the test data, and basically what we have here is an overfitted curve or an overfitted model. Such a model has great performance in training, and we believe that we've built an amazing model, but this model performs poorly in real usage. In fact, if you were to be using this model for prediction, you might have been better off with a far simpler model. A simple straight line through this data performs worse in training, but better with test data. An overfitted model in summary has a low training error. The model does very well with training data, but it has a high test error. The model performs poorly with real data it hasn't seen before. An overfitted model can be thought of as a suboptimal choice that was made in the bias-variance tradeoff. An overfitted model is one that has a high variance error, but a low bias error. If you don't understand what exactly these terms mean, never mind; that's exactly what we're about to discuss over the next few minutes. Every machine learning model, no matter what algorithm you choose, has a bias and a variance. A model is said to have a low bias if it makes few assumptions about the underlying data. It's set to have a high bias if it makes more assumptions about the underlying data. For example, a simple linear model assumes that a straight line can represent the relationships in your data. That's an example of a high-bias model. A low-bias model typically works with your data as is. It makes no assumptions about the relationships that exist. This means that training data in a low- bias model is all important and the model parameters count for little. When you have a low-bias model, your model can end up being very very complex. On the other hand, if you have a high-bias model, your model can tend to be overly simple. The model

parameters which account for the assumptions that your model makes are all important. Training data counts for little. Now that we've understood bias in a model, let's move on and understand variance. A high-variance model is one where the model changes significantly when the training data changes. The model makes two assumptions about the underlying data so it varies when the data changes. On the other hand, a low-variance model is where the model doesn't change much when the training data changes. The model is not as sensitive to the training data. High-variance models can end up being overly complex. The model varies too much with changing training data. The model is far too sensitive. On the other hand, low-variance models can end up being overly simple. The model is not very sensitive to training data; it's not really capturing all of the information that's in there. As you can see, bias and variance are at the opposite ends of the spectrum so you have to have the right tradeoff to build a robust model. A complex model is one which has a high-variance error, and a simple model is one which has a high-bias error, and a good model is one which makes the right tradeoff between bias and variance. That of course depends on your data and your use case. There are some models that are known to be high-bias models. These are models with simple parameters and regression models are examples of high-bias algorithms. Regression models tend to make assumptions about the shape of the curve that fits your underlying data. On the other hand there are other machine learning algorithms that are known to be high variance, which have complex parameters. These are decision trees and neural networks. If you want to prevent your model from overfitting on the training data, there are several techniques that you can employ. Regularization, cross-validation, ensemble learning, and dropout. Regularization is a technique that you'd use with your machine learning models to penalize models that are overly complex. You'll tweak the objective function that your model is working with to add an additional penalty if your model is complex. It forces the optimizer under the hood to keep the model simple. Cross-validation is another technique to mitigate overfitting. You'll have distinct training and validation phases for your model. You'll train different models using just the training data, and you will evaluate these models using the validation data. So you'll select that model that does best on the validation data. And this process of selecting the best model on the validation data is referred to as hyperparameter tuning. We'll discuss this in a lot more detail later on in this course. Ensemble learning is another technique that you can use to mitigate overfitting. You will construct several machine learning models and then combine their output to get the final result. Each individual model in this ensemble of learners could be a relatively weak learner. It may not be a great model by itself; however, when you combine many of these weak learners together, it can yield a strong learner a robust model. Dropout is a specialized technique used to mitigate overfitting in deep learning models such as neural networks. Deep learning models such as neural networks consist of layers of interconnected

neurons. Dropout involves intentionally turning off some neuron at random during the training phase of your model. Active neuron in any iteration determines the architecture of your neural network so every iteration during training thus has a subtly different architecture.

Accuracy, Precision, and Recall

In this clip, we'll talk about measures that you can use to evaluate classification models, specifically accuracy, precision, and recall. Machine learning techniques have been around for a while; however, in recent times the most ground-breaking applications of ML have been two classification problems: image recognition, medical diagnosis. All of these are examples of classification models. The most commonly used technique to evaluate classification models is accuracy. This is where we compare predicted values from your model and actual labels. If more predicted values match actual values, that results in a higher accuracy. Now high accuracy sounds great, but it's not necessary that a model with high accuracy is a good one. Depending on what you plan to do with this machine learning model, it's possible that an algorithm with high accuracy might still be a poor machine learning model whose predictions are useless. Let's say you were building a machine learning model to detect cancer. Now this is a all-is-well binary classifier. It takes in medical reports as its input, and it always classifies these medical reports as normal, no cancer. Let's say that the cancer type that you're trying to detect is extremely rare and not present in 99.99 % of the population. The accuracy for such a model might be huge, but would you consider this to be a good model? Not really. It's quite possible that some labels such as the presence of cancer might be more common or rarer than others. In such situations, the dataset that you're working with is set to be skewed, and when you're working with skewed data, accuracy is a poor evaluation metric for your model. Let's assume a simple binary classifier, our cancer detector. You can plot its predictions using the confusion matrix. Along the columns you have the predicted labels from our model, where a cancer was detected or not. Along the rows, you have the actual label from our data where the cancer was actually present or not. We've run this model on test data and filled in the cells of this matrix based on predicted labels and actual labels. Now this top cell on the left here is comprised of two positives where the actual label is equal to the predicted label, and cancer was detected. The cell at the bottom left here are the false positives. There was no cancer, but our model basically said there was cancer. The actual label is not equal to the predicted label. The cell at the bottom right here is comprised of true negatives. The individual did not have cancer; our model predicted no cancer, actual label equals predicted label. And finally at the top right here we have the false negatives. Here the individual actually had cancer, but our model was unable to detect it. If you were to calculate the accuracy

of this model, you'd only focus your attention on the cells where the actual label was equal to the predicted label. With the instances that you see here on screen in this confusion matrix, the accuracy of this model is 99.12 %. Now this accuracy value makes it seems like this model is an amazing one. This means that the classifier gets it right 99.12 % of the time, but is it really a good model? Think of all of these individuals here who are probably taking treatment for cancer when it's not really required. But, what about all of these individuals here in this other cell? Cancer has not been detected; no treatment has been prescribed for these individuals. It's pretty clear that accuracy in this case is not really a good metric to evaluate whether this model performs well. We need something different and one of those other measures is precision. The precision of a model is basically the accuracy of your classifier when it flags cancer, or in more general terms, it is the accuracy of your model when it makes a positive identification. If you calculate how many of the positive identifications were correct for this model, you'll get a value of 66.67 %, and with precision we can immediately see that our model isn't looking that good. A precision value of 66.67 % essentially means that one in three cancer diagnoses is incorrect. Another metric that you could use to evaluate this model is recall. Recall is the accuracy of your classifier when cancer is actually present. How many of the positive identifications in the underlying data was your model able to catch? The recall for this particular model is 71.42 %. This means that our model missed two out of seven cancer cases.

The ROC Curve

In this clip, we'll discuss the ROC curve and how it can help us find the right tradeoff between precision and recall. Let's understand how a logistic regression classifier works. It tries to fit an S-curve on your data, and you have this vertical line here that indicates a threshold. On the X axis you have a probability score for whether cancer exists or not. If the probability output of the model is below the threshold, you predict no cancer; if it's above the threshold you predict cancer. Now let's say you were to set this threshold of probability score equal to 1. What you have is an Always Negative classifier. Your classifier always predicts no cancer as shown by the green sense here. For such a model it's pretty clear that the recall of this model is equal to 0. It hasn't identified any cases of cancer. The precision is infinite. Whatever it has identified it has got right. The classifier is way too conservative. Let's say you were to plot this on a graph. On the X axis you would have he conservativeness of your decision threshold, and on the Y axis you have the precision score. It's pretty clear that the more conservative classifiers will have higher precision scores, and an Always Negative classifier will have infinite precision. It hasn't made any positive identifications so whatever positive identification that it did make is correct, or you could have a

classifier at the other extreme. Threshold is equal to 0; this is an Always Positive classifier, which always says cancer is present. The recall of such a model is 100%. It will detect all cases of cancer; however, the precision for this classifier will be really low, only about 13.7 % in our example. This classifier is not conservative enough. It's too quick to predict cancer. So if you plot recall on the Y axis and the conservativeness of your decision threshold on the X axis, we'll get a recall curve that looks like this, the inverted S. The less conservative your decision threshold, the better the recall of your model. So it's pretty clear here that there is a real tradeoff between precision and recall. Higher-precision models will have low recall; lower-precision models will have high recall. Let's plot precision versus recall. Recall on the X axis, precision on the Y axis, and you'll get a curve that looks like this. So how do you make the right tradeoff between precision and recall? There are several techniques available. One way to do this is to use the ROC curve. You'll plot the false-positive rate on the X axis and the true-positive rate on the Y axis. Now obviously for our classifier model, we want the true-positive rate, that is the positive identifications made by our model that were correct to be as high as possible. On the other hand, the false-positive rate, the positive identifications that were wrong should be as low as possible. You'll train a number of different models and plot the true-positive rate versus the false-positive rate for each of these models, and the curve that you get is the ROC curve. Here ROC stands for Receiver Operating Characteristic. Once you've chosen your machine learning algorithm, we'll perform something called hyperparameter tuning. Maybe you'll choose different values of P threshold to figure out the true-positive rate versus false-positive rate for each P threshold. Once you've plotted all of these points on this graph, you'll fit an ROC curve for different values of P threshold, and your curve will look something like this. The threshold value that you'll finally go with is the one at the top left corner of this curve. Why? Because this maximizes the true-positive rate and minimizes the false-positive rate.

Preparing and Processing Data

In this demo, we'll build and train a machine learning model to perform classification and then evaluate that model using accuracy, precision, and recall. Here we are on our Datasets page in Azure Machine Learning Studio. Click on the New icon at the bottom left and let's upload from the local machine the dataset that we'll use to train our machine learning model. Click on the Choose File button here in this dialog. The data that we're going to be working with is the breast-cancer dataset available here at this URL. This contains a bunch of information about different women, and we'll use that information to train a model to predict whether an individual has breast cancer or not. Having uploaded the dataset successfully, let's head over to the Experiments

page to create a new experiment. This is a new ML model and that requires a new experiment. Once again, we'll start off with a blank experiment where we'll fill in the details of our machine learning workflow. I'm going to call this experiment breast cancer detection. As before, we'll start off with the data that we're going to use to train our ML model under Saved Datasets, the dataset that we just uploaded, breast- cancer.csv is available. Drag this on to the center pane. We haven't looked at this data yet. Let's right-click, go to dataset, and Visualize and visualize the information that's available. This is a fairly small dataset with just 286 rows and 10 columns. The class columns here are the labels that we'll try and get our trained model to predict, whether breast cancer recurred or not. You can see that most of the cases in our dataset have no recurrence and there are a few cases where the cancer did recur. Exploring linear relationships in data is possible in Azure ML Studio as well. Let's compare the class of recurrence or no recurrence with the Age. You can see for different categories of age whether the cancer recurred or not. If you're interested in exploring this dataset further, you can perform other comparative analysis as well in this dialog. For now, let's go on with building our machine learning model. Let's go to Data Format Conversions and convert this cancer.csv file to a dataset. Pass in the output of the CSV file to the Convert to Dataset node. We'll now perform some preprocessing on this data and for that we'll write some Python code within Azure ML Studio. If you expand the Python Language Modules option, you'll see an option to execute a Python script to clean and process your data. Drag that on to the main pane and connect it to the output of Convert to Dataset. With the Python script node selected, you can open up the Properties panel, which will give you an editor window where you can write your Python script. Let's expand this editor window to full screen so that we can see the code that we write more clearly. You can see that the script is prepopulated with the transformation function that is invoked. The azureml_main function that you see here is the entry point for the script and that should be included. There's a bunch of documentation walking you through how you use this script. Let's go ahead and get rid of those comments so that we can focus on the code that we're about to write to transform this data. We get the input in dataframe1. There are bunch of columns in our data that are categorically values. Here are all of the column names. I'm going to run a for loop through each of these columns and convert these to type category. I access each column, use the astype function to make it a categorical variable. I've made the changes to dataframe1 and this updated dataframe is what I return from this transformation function. Even though you can use Azure ML Studio without writing any code, you can see how easy it is to include a little bit of Python to clean and transform your data. Back in the Properties panel you can specify the Python version and runtime where you want this code to be executed, Anaconda 4.0 /Python 3.5 is the choice is that I've made. Now that we have a number of processing steps in our workflow, let's go ahead and run all of these to get the

transformed data. If you want to see the results of executing this little Python script, we can right-click on the Python Script node and choose Visualize. Here is the original dataset that we were working on. You can see that the class labels indicating whether cancer recurred or not are now represented using 01 numeric values rather than 3s. Earlier, the age data for the records in our dataset were expressed in the form of age ranges. Converting this column to categorical datatype has expressed these age ranges in the form of numeric labels. Now that I have the columns expressed in the form I want, I'm going to use the data transformation and manipulation option that I have here to edit the metadata for all of the columns so that Azure ML can identify them as categorical values. Drag this onto the main pane and connect this to the output of Execute Python Script. Go to the Properties for Edit Metadata and launch the column selector. Instead of manually specifying the columns for which I want to edit metadata, I'm going to specify the rules which will allow me to select columns. I'm going to specify rules that apply to all columns, and I want to include those columns where the column type of the column is a numeric type. I'm going to edit the metadata for all numeric columns in our dataset, which if you go to the By Name option, you'll see is basically all of the columns in our dataset. All of the columns have been selected; they are all numeric, except the selected columns, and let's go back to the Properties panel and change all of these columns to be categorical data. Make them all categorical so that Azure ML can work with them. In order to execute Edit Metadata, I'm going to select and right-click on this node and choose the Run selected option, which will execute only that node. Once the execution has been completed, you'll see that all of the columns are now categorically columns in our dataset. If you visualize the results of this particular node, we'll see that class is a categorical feature as is age and all of the other columns are categorically features as well. I'll now perform a bunch of processing on this dataset, and I want to store this dataset as a CSV file. I'll go to Data Format Conversions and convert this to a CSV. Drag this node onto the center pane, select it, and choose the Run selected option to execute this node. Once we've converted this transformed dataset into CSV format, we can right-click on the Convert to CSV node and save this result as a dataset, and I'm going to name this dataset cancer_data. This is the cleaned and transformed version of the original dataset. Once the output has been saved as cancer_data, you can head over to datasets and see that a new dataset is available in your Datasets link. We'll use this cleaned and transformed dataset later on when we perform hyperparameter tuning on our model.

Building Training and Evaluating a Classification Model

Here we are back in our experiment where we're going to build a classification model to detect whether cancer has recurred or not. Go to Data Transformation and within Data Transformation under Sample and Split, and select the Split Data processing node that will allow us to split our dataset into training data that we'll use to build the classification model and test data that we'll use to evaluate the classification model. Connect the output of Edit Metadata to the Split Data node. The edited and cleaned data is what we'll pass in to train our model. I'm going to specify a split of 80/20 with eighty-percent of the data for training and 20% to test our model. Once this particular node in our machine learning workflow has been executed, you'll see that there are two outputs to Split Data. Results dataset2 is the test data and Results dataset1 is the training data. You can select and right-click any of these datasets and visualize all of the records that are present in there. You can see that there are 229 rules in this dataset. This is 80% of the original data that we started off with. With all of the data set up, we're now ready to select machine learning algorithm using the Machine Learning drop-down and train our model. Go to Initialize Model and the kind of model that you want to train is a classification model, and you'll see that there are many options here, as you might imagine. I'm going to pick the simplest model here. This is the Two-Class Logistic Regression. The Two-Class refers to the fact that there are two possible output categories. The output could be cancer recurrence was detected or not. With our ML algorithm chosen, I'm now going to train this machine learning model. Under Train we have the Train Model node that we can drag and drop to the center of the screen. Now training the model requires two bits of information. One is where we specify the machine learning algorithm, that is the Two-Class Logistic Regression, and the other where we specify the training data. So we have connections from the ML algorithm, as well as the training dataset to the Train Model node. Now let's go ahead and launch the column selector so we can tell the model what it has to predict. We want to predict the output class that is whether cancer is detected or not, using all of the other features. Select the Train Model node and use the Run option at the bottom, Run selected, to start the training process. Now this might take about 30 seconds to a minute, wait for training to complete, and then you can go ahead and visualize this trained model. Here are on top are all of the configuration parameters that your logistic regression classifier used. These are the defaults config settings. If you scroll to the bottom, you'll find the weights associated with each of the input features. Logistic regression tries to fit an S-curve on the underlying data and every feature like in linear regression is associated with a weight. Now that we have a fully trained model, we need to score our model to know whether it's a good one or not. Go to Score under Machine Learning and there is the Score Model node that you can drag on to the main pane. We want to evaluate our fully trained model so connect the output of the Train Model to Score Model, and for this we'll use the test dataset. So connect the second output of the Split Data, the 20% of

the data that we set aside to evaluate our model to the Score Model node. Let's go ahead and run the selected node so that we know how this trained model performs on the test data. Score Model will give us the predictions of our trained model on the test dataset. Go to Scored Dataset and Visualize, and these are the 57 rules in our test dataset, and at the very right you'll see the scored probabilities of our model. Classifier models always output a probability. In the case of a binary classifier like this one, these probabilities represent a model prediction for whether cancer was detected or not. If the probabilities are above a certain threshold, the default threshold is typically 0.5, cancer was detected. If the probability scores are below the 0.5 threshold, cancer was not detected. The scored labels are derived from these probabilities and are the actual predictions from our classifier model. Select the Scored Labels, which is the output of our classifier, and let's compare these scored labels to the actual labels, that is the class column, and this will give us the confusion matrix for our classifier model. This is the confusion matrix with actual labels in the Y axis and the model output on the X axis. You can see that this model did fairly well. Along the main diagonal are all of the predictions that it got right. Out of the 57 records in the test dataset, it got 39 correct predictions, and these are all the wrong predictions from our model. Eighteen predicted outputs on the test data were completely wrong. We've seen the confusion matrix. The next step is to evaluate our model to get the accuracy precision and recall scores. Under Evaluate, choose the Evaluate Model node and drag it onto the main pane. We have the scored model on the test dataset; that's what we want to evaluate. So connect the output of the Score Model to Evaluate Model, and let's go ahead and run all of the steps in this experiment. Wait for about a minute and once evaluation is complete, we can right-click. Go to Evaluation Results and Visualize. You can see that along with numbers there are also some nice visuals that you can use to evaluate your model. This is a graph of true-positive rate on the Y axis versus the false-positive rate on the X axis. This shows very clearly the more cases of cancer that are detected by our model, false positives also tend to go up. There are other tradeoffs that you can visualize as well such as the precision recall tradeoff. Here is what the curve looks like for our dataset. You can see that when the recall values are low, precision tends to be high, and as the recall values improve, precision tends to go down. If you remember, precision is a measure of positive identifications of cancer that were actually correct, and recall is a measure of what proportion of the actual positives were caught by our model, were correctly identified by our model. If you scroll down below, the actual values for accuracy, precision, and recall and even the F1 score of this model is for you. You can see that the accuracy is around 68%, precision at 40%, and recall at 11%. Now this is something that you can manipulate by changing the threshold at which your model categorizes positive identifications versus negative identifications. Let's say we make the threshold higher. It's now at 0.68 instead of the default 0.5. We can see that the

accuracy has gone up as has the precision. There are fewer false positive cancer cases identified, but the recall has rarely fallen. We're not really identifying as many cases. As you increase the threshold for a positive identification for cancer, precision rarely goes up, but recall goes down to almost 0. If you reduce the threshold for identifying cancer, you'll detect more cases of cancer so recall goes up, but precision goes down because you'll have more false positives. If you bring the threshold all the way down close to 0, recall will go all the way up to 1. Thus when you build and train your classifier model, you can see very clearly the tradeoff that exists between precision and recall.

Summary

And with this we come to the very end of this module where we discuss common problems that we encounter in data modeling and how we can overcome these. When you're building machine learning models, one of the most common problems that you'll encounter is that of overfitted models. We understood what it means to overfit on the training data. An overfitted model is one which does very well in training, but poorly in the real world when used for predictions. We get an overfitted model when we do not make the right choice in the bias variance tradeoff. We understood what it means to build high-bias and high-variance models and the pros and cons of both. We also discussed several techniques that could be used to mitigate overfitting such as regularization, cross-validation, ensemble learning, and dropout. We then moved on to discussing classifier models and techniques that we could use to evaluate them. We saw that a high-accuracy model is not necessarily a good one. If you have a skewed dataset, you might want to evaluate your model using other metrics such as precision and recall. Choosing between precision and recall is a tradeoff. We understood the use of the ROC curve in making this decision. And finally, we brought all of this together in a hands-on demo on Azure ML Studio where we built a model to detect cancer. In the next module, we'll turn our attention to cross-validation, which we can use to evaluate our model and mitigate overfitting.

Leveraging Different Validation Strategies in Data Modeling

Module Overview

Hi and welcome to this module on Leveraging Different Validation Strategies in Data Modeling. We'll start this module off with a discussion of IID data, data that is independent and identically distributed. We'll see that this is an implicit assumption that is made in most modeling techniques. We'll then discuss how cross-validation can be used to build robust machine learning models. We'll also see a hands-on implementation of cross-validation on the Azure ML Studio. We'll first understand what it means to perform singular cross-validation on your data. We'll then move on to other cross-validation strategies such as the iterative K-fold cross-validation, which divides your training data into folds and every fold but one is used to train your model, one fold is used to validate your model. We'll discuss that this technique is computationally intensive, but robust. We'll then discuss other variants of K-fold cross-validation such as repeated K-fold cross-validation and stratified K-fold cross-validation. Stratified cross-validation splits data so that class values are maintained in their representative proportions. We'll also discuss grouped cross-validation that is a special procedure used to work with grouped data.

Cross-validation in the ML Workflow

We've already briefly discussed cross-validation as a technique that you can use to evaluate your model. Let's see exactly what cross-validation is. This is a model validation technique to assess how the results of a statistical analysis will generalize to an independent dataset, and this is what helps determine how well a module will perform in practice. Cross-validation involves validating our models on data that it hasn't encountered before in the training phase. Before we get further into the discussion of cross-validation, let's discuss the IID assumption. Usually we assume that points in a dataset are independent of each other and identically distributed, that is similar to each other, as well as the statistical properties are concerned. The IID assumption is simple to state, but it's an extremely important one. This is implicit in the training of virtually all machine learning models. All of the cross-validation techniques that we'll discuss in this module, except for one, that is the grouped cross-validation, assumes that we are working with independent and identically distributed data. Early on we had discussed the steps involved in building and training your machine learning model. This is the basic machine learning workflow. We saw that we needed to fit a model on training data. This is the process of training your model and once you have a fully trained model, you need to choose a validation method to evaluate the model. This is where cross-validation fits in. We'll use cross-validation to evaluate our model. We'll examine the fit and update the model if you're not satisfied with its performance. Now that you understand

where exactly cross-validation lies in this machine learning workflow, we can move on to understanding techniques that can be used for cross-validation.

Singular Cross-validation

In order to understand cross-validation, let's start off with the simplest possible technique, that is singular cross-validation. Here is a quick overview of what it takes to evaluate model performance. Once again, we'll dive into each component of this diagram in detail. You'll feed in all of your different candidate models, you'll perform cross- validation on them to find the candidate model that fits best, one which has the best parameters. Cross-validation is what you'll use to refine your original model parameters to find the best model. Once you have the best parameters for your model, you'll then build your final model using these parameters. Cross-validation will just help you find the best candidate model. You'll then need to retrain this model using your training data, and once you have a fully trained model, you'll then test this model on the test data set to get the final evaluation. With this big picture in mind, let's talk about what exactly is singular cross-validation. Now if you have just the training data and test data, you're essentially performing no validation at all. You have just the one candidate model. You'll train it using the training data and you'll sanity-check it using the test data. That's essentially what we've done so far. If you're evaluating N different models, you'll simply run N training phases and then N test phases. There is no explicit validation here. You're simply training your models and then testing it. If you're not performing validation on your model, you might overfit on the Test Set. When you choose the best candidate model on Test Set, that leads this kind of overfitting. And this kind of overfitting is obviously likely to occur when we have just two sets of data that we work with, the training data and the test data. To mitigate overfitting on the test set, you'll need validation, and singular cross-validation is the simplest kind of validation that you can perform. Now you can have multiple candidate models, and you can evaluate these models using hyperparameter tuning on the validation data. Singular cross-validation involves the use of three sets of data. You have the training data, the validation data, and the test data. With singular cross-validation, you'll train your model using the training data; nothing really changes there, but you'll evaluate your model using validation data. So if you have N candidate models that you want to choose from, you'll run N training processes and N validation processes, but just one S process. Cross-validation basically involves carving out a separate validation set of data points, and using this validation set to evaluate your candidate models. So your original data will now be split into three sets; training, validation, and test so here is the basic process involved in singular cross-validation. You have your entire dataset represented by all data, which is split into training data, validation data, and

test data. You have a number of different models that you want to evaluate. You'll train these models using the training data, and you'll evaluate them using the validation data. The process of evaluating these models is hyperparameter tuning, and once you've found the best possible model, you'll run it on the test data to see how it performs.

Cross-validation Using Azure ML Studio

In this demo we'll see how we can perform cross-validation on our regression model to see that it's a robust one. Cross-validation, as you know, is a model validation technique for assessing how the results of any statistical analysis, that is our machine learning model, generalizes to an independent dataset that the model hasn't encountered before. Here we are in the Experiments tab where we have the two experiments that we've created. I'm going to click through the KCHousePricePrediction experiment. In the left navigation pane under the Machine Learning category and under the sub category Evaluate, you have the option to cross-validate your model. Drag this node onto the center pane and position it close to the model that you want to cross-validate so that it's easy to work with. Connect this Linear Regression model to Cross Validate Model. Cross-validation involves training several different machine learning models on different subsets of your input dataset. So take the output of clean missing data, that is your final dataset, and connect that as an input to Cross Validate Model. To the Cross Validate Model mode you need to specify the column that you want your model to predict. Launch the column selector. We want to predict the price by using all of the other column values as features to train our model. So move price over to the selected column, click on Accept, and go ahead and run this Cross Validate Model node. Cross-validation involves building and training several different ML models so this might take a little longer. You'll need to be a little patient. Right-click on Cross Validate Model, and go to Evaluation results by fold, and click the Visualize option. You can see that there are a total of 12 rows here. The default cross-validation used by Azure ML Data Studio is 10-fold cross-validation. We've subdivided the dataset into 10 different folds and trained 10 different models using nine folds for training and one fold to test the model. Ten out of 12 rows are fold-wise evaluation results and the remaining two rows are aggregations such as mean and median of results. Here are the actual results of cross-validation. You can see the fold number, the number of examples in the fold, the kind of model that was trained, all of these are linear regressors, and if you scroll over to the very right, you'll see the R-squared scored, which is the evaluation result for each model calculated on the test fold. You can see that the mean value for the coefficient of determination or R-squared is about 0.6, the median is about 0.65. If you scroll down to the bottom on the right pane, you'll be able to view a frequency distribution of the R-squared scores

as well. You can see that except for one particular run of the model, all of the R-squared scores are at around 0.6. The distribution of the R-squared scores might be better viewed as a box plot. Scroll over to the left and select the box plot option, and you can see that except for one outlier, all of the scores are clustered very close together between 0.6 and 0.65. Cross-validation has shown us that this is a fairly robust model and the R-squared score that we got for our trained model wasn't a fluke or by chance. Let's head back to the main experiment page and use the Save icon here at the bottom to save this experiment. We've also trained a linear regression model using a Python notebook. Head over to Notebooks and click through to KCHousePricePrediction and let's see how we can cross-validate this model that we built using scikit-learn. I'm going to scroll down here in this notebook and comment out the code that exists in one cell. This is the code that converts the housing prices dataframe into a dataset. I don't want this executed again. Once this code has been commented out, go to the Cell drop-down here and select Run All cells to execute all of the code that exists in this notebook. The last step here in this notebook was where we built and trained our linear regression model using the linear regression scikit-learn estimator object, and we scored this model on the test data and got an R-squared score of around 0.65. Now let's cross-validate this model using scikit-learn. From sklearn.model_selection import the cross-validation score. Now there are different metrics that you can use to evaluate your model. Sklearn.metrics .SCORERS .keys will list all of the options that you have. These evaluation metrics apply to all kinds of models, clustering models, classification models, regression models. Here are some evaluation metrics that you can use for classification models; accuracy, precision, and recall. Ours is a regression model though, which is why I'm going to use the R-squared evaluation metric. I've instantiated a linear regression estimator object. I'll pass the estimator to cross-validation score along with the X features and Y labels used to train the model. Cv=5 indicates that this is five-fold cross-validation. Scoring= r2 is the evaluation metric that I've chosen for my regression model. This will train five different ML models on different subsets of your data and all of the evaluation results will be present in the scores variable. You can see that we have five scores here for each of the five folds. These correspond to the five folds of test data on which these models were evaluated. All of the scores are around 0.65, indicating that this linear regression is a robust one.

K-fold Cross-validation and Variants

Now that we have a handle on what exactly cross-validation is, in this clip we'll study K-fold cross-validation and its variants. Let's first define and understand K-fold cross-validation. For each candidate model, you need to repeatedly train and validate using different subsets of the training

data. Now the disadvantage here is because of the number of subsets that you might use to train your data, this ends up being much more computationally intensive, but this is a very robust process and it does not waste data. Let's understand K-fold cross-validation visually. You have your entire dataset represented by all data and it's divided into training data and test data. Now consider only the training data and split it into folds. The number of folds is up to you and that is what K represents here in K-fold. Observe that just the training data has been divided into folds. Now for each split of your data, one fold will be used to validate your model, the remaining folds will be used to train your model. In split 1 here, fold 1 is our validation data; fold 2, 3, 4, and 5 will be used to train our model. In split 5, fold 5 is the validation data; fold 1 through 4 is used to train our model. If you have any candidate models, all of your models will be trained on all of these splits, which is why K-fold cross-validation is so computationally intensive. You'll find the average performance of your model across all of these splits and use that average performance to choose the best candidate model. The best candidate model is then what you'll run on test data. So in K-fold cross-validation for N candidate models and K folds, you'll run N multiplied by K training processes, and N validation processes, but you'll just have one test process on the whole. A technique that you can use to improve the performance of cross-validation is the shuffle and split. Shuffle the points in the dataset before splitting it into training and test splits. This helps ensure that test points are picked from throughout the dataset. Now that we've understood the basic K-fold cross-validation technique, there are several variants of K-fold that you can use. The repeated K-fold as its name suggests simply repeats K-fold N type. So you have the same K-fold split of your data and each split will be repeated N times to cross-validate your model. The leave-one-out variant of the K-fold is a simple cross-validation. Every training set is created by taking all samples except one, and the test set is just that single sample, and the last variant that we'll discuss here is leave-P-out, which is simply an extension leave-one-out. Leave-P-out creates all possible training and test set combinations by removing P samples from the complete set. Before we move on to discussing other cross-validation techniques, you need to understand the concept of class and group. Class refers to the output labels or categories associated with your data; male or female, true or false. Group on the other hand refers to measurement groups for your records such as before and after, this week, the next week, yesterday, today; all of these are examples of group. Now that we've understood class, as well as group, you should know that K-fold cross-validation does not take into account either the group of your data or class while splitting your data, and this can be a drawback. There is one variant of K-fold called stratified K-fold that takes into account the class or category of data. The stratified K-fold ensures that every fold has a representation of different classes. Let's say your original data is categorized into male and female. Each fold will have approximately representative mix of male and female as exists in the

entire dataset. So if your original dataset has a 100 records with 80 male and 20 female records and let's say you divide into 10 folds, every fold will have 8 males and 2 females approximately. We discussed both class and group earlier. Stratified cross-validation techniques account for class, but not for group. When you're working with group data, the samples present in your dataset are not independent of each other and must be identified as related to each other using group identifiers. All medical records that are associated with one patient is an example of a group. When you're working with real-world scenarios, you'll find that group data is very, very common. For example, before and after tests. What weight was the patient before he started treatment, after he completed treatment? Another example of group data could be observations that you receive from sensors; some of the sensors may have known biases, or multiple patients and multiple samples per patient. All of these are examples of group data that you might have to work with, and when you have group data, the IID Assumption that we made up front no longer holds. We've spoken earlier of the fact that this is an implicit assumption usually made in most kinds of modeling including machine learning models. When you have group data, you can no longer assume the data points or records are independent nor are they identically distributed. They are associated with groups. So when you're working with group data, you'll need to use special cross-validation procedures such as the group K-fold. These cross-validation procedures need to be aware of the group to which each record belongs and they need to ensure that each group is either entirely in the training data or entirely in the validation data. These cross-validation procedures split your data into folds and they ensure that group IDs do not cross fold boundaries.

K-fold Cross-validation in scikit-learn

Now there are different techniques that you can use to split your dataset for cross-validation. K-fold cross-validation is what we'll study here in this demo. I'm going to be writing code here on a brand-new Azure notebook called KFoldCrossValidation. Set up the import statements for all of the Python libraries that we need. Specifically import the K-fold model from sklearn.model_selection. Instantiate the Azure ML workspace. This references the current workspace which holds this notebook, and within this workspace we have the houseprice_data dataset that we had saved earlier, and that's what we will do access. This is the dataset that we saved earlier on in this course after performing all of the data cleaning and preparation operations. Convert this to a Pandas DataFrame and that's what we'll work with. The housing price values that are our Y variables are available in the form of a Pandas 2D object. I want to convert this to a NumPy array. This I do by accessing the price column in my dataframe and converting it to a floating point data type and converting this Pandas column into a NumPy array,

and then reshaping the array to be a two-dimensional array. The Y variables should be in the form of a 2D array, because that's the format that our linear regression model that we'll build at the very end requires the data to be. Let's save all of the features that we'll use to train our ML model, that is all of the columns other than the price column and store it in the X variable, and I'm going to convert X to a NumPy array as well. Let's take a look at the shape of the resulting X NumPy arrays. There are about 21, 000 records and 15 columns in our training features, that is our X data, and the shape of the Y variable is thanks to the reshape that we performed on our NumPy array. Instead of the 1D vector that it was earlier, it's a 2D array. In order to perform cross-validation, you need to split your data into fours and one way to do it is to use the KFold object in scikit-learn. That's why the number of splits here is equal to 3, and once you have the K-fold object, call the split function on that object and pass in the data that you want to split in the form of an array. N splits is equal to 3 here. The K-fold object has split the data into three subsets and for every split, two of the subsets will be used to train the model and one subset will be used to evaluate the model. I'm going to print out these results in a slightly different way so that things are a little clearer. Initialize a variable i is equal to 1, and iterate over all of the splits in the data. Here is a for loop that iterates over each split. Each split comprises of training data and test data and we print out the train and test data for each split here for every value of i. I will go from 1 all the way through to 3. The original data was split into three subsets. Two subsets are used to train our model, one subset will be used to test our model, and the subsets change for each of the splits. This might become clearer if you take a look at the length of the training and test data for the last subset. The total size of the dataset is 21611. The training data is about 14, 400, two-thirds of the original data, and the test data is about 7200, one-third of the original data. Now the number of splits that you want to use to cross-validate your model is up to you. You can instantiate the K-fold object with N splits equal to 4, and let's print out the training and test data using a for loop. When you see the results you see that we have four splits of the data. Seventy-five percent of the data in each split will be used to train the model, twenty-five percent to test the model, and you can confirm this by printing out the length of the original dataset that is X, the length of the training data, and the test data, and you can see that the split is 75/25. Using the KFold object you can split the data into however many folds that you want, such as N splits = 15. We'll get 15 different sets of training and test data. If you use these splits, you'll train 15 different models. Fourteen by 15 of the datasets will be used to train your model, one-fifteenth to test your model. Let's use the KFold object to split our data into two subsets and we'll use these two subsets to cross-validate our model. We'll store the list of splits of our data in this indices list variable. You can see that the indices list is a list of tuples where every tuple has the training data and the test data. Let's take a look at the first tuple here in this indices list at index 0. This is one tuple where

half of the data is used to train our model and half to test our model. We'll use these folds to train two separate linear regression models. The variables for the suffix 1 form one split, which will be used to train one model, and variables which have the suffix two are the second split, which will be used to train the second model. So for the first model, we'll use the first half of the data to train the model and the second half to test the model and we'll switch this around for the second model. The second half to train the model and the first half to test the model. Let's take a look at the shape of the training data and the Y labels for the first model, and here is the test data and the Y labels for the first model. When you take a look at the training data for the second model, you can see that its size corresponds to the test data that we'll use for the first model. Similarly the test data for the second model corresponds to the training data that we'll use to build the first model. We're now ready to start training our two models, instantiate a linear regression estimator object, and call fit on x_train1 and y_train1. Once we have a fully trained model, let's call this model on the test data, x_test1 and y_test1, and you can see that the R-squared score is about 0.645. Now let's train our second regression model, instantiate the linear regression estimator object, and call fit on x_train2 and y_train2, and let's call this regression model on x_test2 and y_test2, and the score here is about 0.644. This regression model seems fairly robust. The scores on the two cross-validation folds are almost the same, but the way we built this cross-validation was rather tedious. A better way is to use the cross_val_score function. Pass in the regression model that you want to train, pass in the X features and Y variables, and specify cv=2. If you look at the resulting R-squared scores of cross_val_score, you'll find that they match our two models exactly. Under the hood, cross_val_score used the K-fold split algorithm to split the data into subsets and cross-validate our model.

Repeated K-fold Cross-validation in scikit-learn

In this demo we'll see how we can perform a repeated K-fold cross-validation in scikit-learn. Here we are on a new Azure notebook. Go ahead and set up the import statements for all of the packages that you need. Make sure that you import the repeated K-fold object from sklearn_model_selection. We'll work with the same houseprice dataset that we are familiar with, convert it to a dataframe, reshape the price column to a floating point two-dimensional NumPy array, and assign it to the variable Y, and convert the X features to an NumPy array as well. The repeated K-fold is a variation of the K-fold cross-validation where every split is repeated the number of times that you specify. Here are number of splits will be equal to two, and the number of times each split is repeated is also equal to 2. If you take a look at the resulting splits, you'll see that because N splits = 2, at each split half of the data will be used to train the model and the

other half to test our model, and each split is repeated twice. The repeated K-fold can be used to build a more robust model, because it ensures that any result that we get from a certain split of the data is not a fluke or by chance. Let's take a look at the training and test data for each split using the for loops that we can see things more clearly. We can see that every split is repeated twice. This is essentially the same split of the data that you get when you use K-fold repeated N times; here N = 2. Let's take a look at the length of the original data and our training and test data for one split, and you can see that our training and test datasets are almost the same. We generated two folds from our original data, because n_splits = 2. Let's specify n_splits = 2 and n_repeats = 3. We have two splits of our data and every split is repeated three times. When we say n_splits = 2, we see that 50% of the data is used to train a model and fifty-percent to test our model. This you can confirm by taking a look at the length of the original data and the training and test data. Depending on how you want to cross-validate your model, you can specify the number of splits and number of repeats. Both are three here, that using this configuration we have nine different models that we can train and evaluate. Each model will be trained using two-thirds of the data in the original dataset and one-third of the data will be used to evaluate the model. Now let's set up a repeated K-fold with n_splits = 2 and n_repeats = 1. This is just ordinary K-fold. Let's now access the training and test data in each of these splits and train two separate linear regression models. Each time we'll train our model with half the data and test the model using the other half of our data. Let's instantiate our first linear regression model and call fit on x_train1 and y_train1, and let's call this model on x_test1 and y_test1. The R-squared score here is about 645, that's to be expected. Let's now train the model using x_train2 and y_train2, and test using x_test2 and y_test2. We know this model is a robust one. The R-squared score is once again around 0.65. This tedious process of training and evaluating multiple models using cross-validation is made simpler using the cross_val_score function. Pass in the regression estimator object, the X and Y values, and cv here is the object of the kind of cross-validation that we want to perform, repeated K-fold. You can configure the cv parameter based on the kind of cross-validation that you want to do. Here you can see that our repeated K-fold gives us the same scores for the underlying model.

Stratified K-fold Cross-validation in scikit-learn

In this demo we'll see how we can use the stratified K-fold technique to cross-validate your classification model. On a brand-new Azure notebook, import the Python libraries that you need, import the stratified K-fold object from sklearn_model_selection. Instantiate the Azure workspace and load in the cancer data that we had saved as a dataset earlier in this workspace and convert

it to a dataframe. Here is a quick refresher of the data that we are working with. This data has already been pre-processed and all of the columns have been converted to categorical values. The Y variable or what we want our model to predict is the class column where the cancer recurred or not. All of the other columns in our dataset from the X variables that we used to train our model. Drop the class column and assign the rest to X. This dataset contains a total of 286 records. Now our dataset contains records that can be classified into two categories, whether cancer recurred or not. When you use the stratified K-fold algorithm, it'll split the dataset into folds where the folds are made by preserving the percentage of samples for each class. N_splits = 5 so we'll get a total of 5 splits of data, and every split will have four-fifths of the data for training and one-fifth to evaluate the model. To keep the output simple, I'm going to print out just 10 records from the training and test data. The result will be five splits of training, as well as test data. In each fold, the percentage of cancerous and non-cancerous records will be preserved. Let's use the stratified K-fold object to get two splits of our data and let's assign these splits to the indices list. We'll use these two splits to train two classification models using logistic regression. Here is the training and test data for the first model, and here is the training and test data for the second model. We'll train our first classifier using 142 records for training, and we'll use 144 records to test that classifier. We'll reverse the training and test data for the second classifier model. We'll use 144 records to train the model, and 142 records to test the model. Instantiate the logistic regression estimator object from scikit-learn to perform logistic regression. Solver equal to liblinear is simply the optimization method that logistic regression uses under the hood. C stands for the regularization that we want to apply to our model. Smaller values of C implies stronger regularization. This is a lightly regularized model called fit on x_train1 y_train1, and score this model on x_test1 and y_test1. The default score for classification models is the accuracy of the model, and the accuracy here is 73.6 %. Let's now train our second classification model. Once again, we'll use the logistic regression estimator object called fit on x_train2 y_train2, and score the trained model on x_test2 and y_test2. The accuracy of this model is 56.3 %. That's pretty low considering that this is a binary classifier. Fifty-percent is the accuracy that you'd expect if you use if pure chance to predict the output; 56% isn't that great. Instantiate the estimator object that you want to train, and let's use the cross_val_score to perform cross-validation. Here cv=2 and you should know that this function, if you specify a model for binary classification, by default uses stratified K-fold under the hood. For other models, the default is the K-fold cross-validation. For any classifier model it is the stratified K-fold, and if we take a look at the results, you'll see that this bears out. We get the same accuracy scores as when we used stratified K-fold and individually trained two separate ML models. If you want to score your models using a metric other than accuracy, you can specify any of the metrics here in this list.

Let's use precision and recall to score our model. The cross-validate function in scikit-learn allows you to specify a scoring parameter. Here we pass in our scoring array, which includes precision and recall to score the different cross-validator models. The return_train_score=True parameter that we pass in will also return the scores for the training data. The resulting scores returned by this function will give you the precision and recall scores on the training data, as well as the test data.

Group K-fold in scikit-learn

In this demo we'll see how we can perform cross-validation in scikit-learn using group K-fold. We start off in our Azure ML workspace. Under Datasets we need to upload a new dataset to work with group K-fold. Click on the New button here on the Datasets page and select a local file, which we'll upload to the Azure Cloud. Choose File on this dialog. This is the weight_loss.csv file, a made-up dataset that I've created to explain group K-fold. Click the checkmark on this dialog, and this new dataset will be uploaded to the Azure Cloud within your workspace. Now we can head over to Notebooks and get started with a new notebook. Select the Python 3 column for this notebook. All of our code has been written using Python 3. I'm going to call this the GroupKFold notebook. Set up import statements for all of the Python libraries that we need; specifically we need GroupKFold from sklearn.model_selection. Instantiate an Azure ML workspace as we do normally and access the weight_loss.csv dataset that we just uploaded. Let's take a look at this data in the weight_data dataframe, and let's discuss why we need the GroupKFold algorithm to split this data into subsets before we use it to train a model. This data corresponds to four different patients who've gone in for some kind of weight-loss treatment. Notice that we have patient ID 1 here. We have the weight of the patient before the treatment, and the weight of the patient after the treatment. Here is another patient, one with ID 2. We have the weight of the patient before the treatment and after the treatment. Every patient is a group, and this group logically belongs together in the same fold. And this is what the GroupKFold split of our data will ensure. GroupKFold will ensure that all of the data for a single patient, let's say it's patient 1, is present in one fold, and it's not split across folds. Let's preprocess this data a little bit. We'll convert the treatment column to numeric columns, instantiate a label encoder, and call fit_transform on the treatment column. The treatment column consists of categorical values that are strings. The label encoder will convert this to numeric values. The label 1 here indicates before treatment and 0 after treatment. Now to understand how GroupKFold splits the data, let's create our X and Y variables. We won't actually fit our machine learning model because this is made-up data, but we'll see how GroupKFold splits our data into subsets. Let's assume that the X data is

just a treatment, and the weight column. We'll drop the height and patient ID. The Y data is simply the height of the individual, and the groups we specify by the patient ID column. All of the data related to one patient is one group in our dataset, and these groups have to be preserved and our dataset is split into folds. Let's now instantiate a GroupKFold object and specify n_splits=4. There will be four splits in the resulting data. When you call GroupKFold.split on the data, we need to specify X and Y, as well as the groups in the dataset. We'll now print out all of the training and test data in our four splits so that you can see how the folds have been generated. Here is the original dataframe and all of the records that are in our training data. Observe that patients 1, 2, and 3 are all present in the training data, and the patient 4 is now present in the test data. All of the data that corresponds to a particular patient, that is a group, is preserved in a fold. Here is another split where the patients one, two and four are in the training data, and the patient with ID 3 is part of the test data. Let's see another example of the GroupKFold split where n_splits=2. I'll now split our X and Y data based on the patient ID, which is our group, and here is the result of this split. Half the records that make up the training data of the first split belong to patients with ID 1 and 3, and the test data for the first split comprises of patients with ID 2 and 4. Thus you can see that the GroupKFold algorithm preserves groups within a fold.

Summary

And this demo gets us to the very end of this module on Leveraging Different Validation Strategies in Data Modeling. We started this module off by discussing what it means to have IID data, data that is independently and identically distributed. We discussed that this was an important implicit assumption in almost all kinds of data modeling. We then moved on to discussing cross-validation techniques to build robust models. We started off with a discussion of singular cross-validation and moved on to other cross-validation variants such as the iterative K-fold cross-validation, the repeated K-fold cross-validation, and the stratified cross-validation. We first understood the concepts of class and group where class refers to the output label associated with every record, male or female, true or false. These are examples of classes. We saw that stratified K-fold cross-validation ensures that every fold has a proportionate representation of different classes. We then discussed grouped data where a group refers to a measurement group, and we saw that group data requires special cross-validation techniques. We then performed hands-on implementations of cross-validation using the Azure ML Studio, as well as the Python scikit-learn library. In the next module, we'll evaluate and refine our models using hyperparameter tuning.

Tuning Hyperparameters Using Cross Validation Scores

Module Overview

Hi and welcome to this module on Tuning Hyperparameters Using Cross-Validation Scores. Now that we understand what exactly cross validation is, we know that we can use cross validation to evaluate our model. In this module we'll discuss hyperparameter tuning. Hyperparameter tuning is essentially a way to design the best model to use for your data and with your data. Hyperparameter tuning is a process where you train many different candidate models and pick the best one using cross-validation scores. We'll first discuss the idea of model parameters versus model hyperparameters and understand the difference between these two. We'll then build a classifier model on our cancer dataset using the decision tree algorithm. We'll then perform hyperparameter tuning on this model using the Azure ML Studio. Before we get any further into this module, let's first discuss what hyperparameters are. These are model configuration properties that define the model and these properties remain constant during the training of the model. I'm going to repeat this significant bit. Training your model does not change the value of hyperparameters. These remain constant and hyperparameters can be thought of as a part of your model's design. When you're working with your machine learning model, there are several bits of data that you use. The first are the model inputs, then we have the model parameters, and finally we'll work with the model hyperparameters. Model inputs are just the training data that you use to train your machine learning model. These are the input data points. Model parameters refer to the trainable or learnable parameters in your model. These are the values that are tweaked during the training process. For example, if you're building a decision tree model, the structure of the decision tree will change based on your training data. Model hyperparameters refer to the configuration of your model itself. These are values that do not change during the training process; they form a part of your model's design such as the depth of the tree, minimum number of samples per node. All of these are hyperparameters for decision trees.

Hyperparameter Tuning

Now that we've understood what hyperparameters are, let's see where hyperparameter tuning fits in as a part of the machine learning workflow. Here is a simplified workflow. We start off with data preparation. This preprocessed data is then used for model development. Once we've developed our model, we'll train and evaluate the model to see if it's the best fit for our data. This is model training and evaluation. Once we've found the best possible model, we'll deploy this model and use this model for predictions, and if you have multiple versions of the model that we have trained using new data, we need to be able to manage those versions. Data preparation and model development is typically performed using machine learning frameworks such as TensorFlow, Keras, XGBoost, and scikit-learn. In this course we used a no-code option available on the Azure Cloud through Azure ML Studio. Once you've developed your model, you'll probably perform distributed training if you're working on a huge dataset. The distributed platform that will train your model and the ML framework that you've used to develop your model need to work closely together. As a data scientist and a developer, your job is to configure the training and hyperparameter tuning of your model. The distributed platform will perform training and hyperparameter tuning of your model. So you need to set it up, the platform will take care of the rest, and when you're working on cloud platforms, you'll find that the remaining steps are taken care of by the distributed platform. There's not much role that you as a developer need to play. This process of model training and evaluation and refining your model using hyperparameter tuning and the deployment of your model requires a very precise handshake between your ML framework and the platform. Remember, training your model is not the same as hyperparameter tuning. Hyperparameter tuning involves finding the best design for your model. Training involves getting this best model to learn from your data.

Decision Trees

In the hands-on demo that we'll perform in this module, we'll perform hyperparameter tuning on decision tree classifiers. So let's get a big picture understanding of how decision trees can be used for classification. Let's say you were looking at a sports person and you wanted to classify him or her as a jockey or a basketball player. Now we know that jockeys tend to be light to meet horse-carrying limits. On the other hand, basketball players tend to be tall, strong, and heavy. Now let's say you know the height and the weight of the individual that you're trying to classify. You would build a tree based on this data. Let's say you were to consider the weight first; we would say greater than 150 pounds, that's a basketball player, less than or equal to 150 pounds, well, let's look at the height. Greater than 6 feet, basketball player, less than 6 feet, jockey. So what you're essentially doing here is fitting the knowledge that you have about both of these

sports into actual rules, and each of these rules involves a threshold. Now it's pretty clear here that the order of the decision variables matter; whether you're considering date first or height, that makes a difference in your tree structure. The training data that you'll use to train your decision tree model will help find these rules and the order of the decision tree variables. A decision tree found in this manner is referred to as CART, Classification And Regression Tree. Decision trees can be used for both classification models, as well as regression models. This decision tree will be built using your training data. Once you have this decision tree, how would you use it for classification? Well, based on the individual that you're trying to classify, you will traverse the tree to find the right node. You'll then return the most frequent label of all of the training data points that lie on that node. There are a number of hyperparameters that go into the design of your decision tree. Let's discuss a few. The first is the splitting strategy. This is the strategy used to choose the split at each node when your tree is being built up. Another hyperparameter that you can set is the max depth of the tree. The depth of your tree will be capped at this max depth and it'll prevent your trees from getting arbitrarily deep. Another hyperparameter is the min samples split, which is the minimum number of samples of data required to split an internal node. Another hyperparameter is the min samples leaf. This is the minimum number of samples or records required to be at a leaf node. If the input records that you feed in to build your tree model are weighted, you have another hyperparameter, the min weight fraction leaf. This is the minimum weighted fraction of the sum of total weights required to be at a leaf node, and finally another hyperparameter is the max features. The number of features to consider when looking for the best split of your tree.

Hyperparameter Tuning a Decision Forest Classifier

In this demo, we'll see how we can use the Azure Machine Learning Studio to perform hyperparameter tuning in order to find the best design for our ML model. We'll create a new Azure experiment for hyperparameter tuning. Click on New and select the Blank experiment template. We'll call this experiment hyperparameter tuning using cross-validation scores. For each model design, ML Studio will perform cross-validation to validate the model and score it. Once we have all of the models scored, we'll then be able to pick the one that performed the best on our dataset. Head over to Saved Datasets and under that we'll look for the cancer data. This is under My Datasets and it's the cleaned version of the breast cancer dataset that we were working on earlier for classification. Drag the cancer data onto the center pane and head over to Data Format Conversions. Let's save this data in the form of a generic CSV file and we'll need to convert it to a dataset before Azure can use it. Drag the Convert to Dataset and connect the output of the

cancer data to this input node. We'll now edit the metadata in order to convert all of the columns to categorical values. Go to Data Transformation and under Manipulation, you'll find that the node which allows you to edit the metadata of columns. Drag this onto the main pane, connect it to the output of Convert to Dataset, and then switch over to the Properties panel and launch the column selector. This will allow us to specify which of the columns we want to make categorical. Using rules that is column names, I'm going to select all of the columns that are present in the dataset. Class, age, menopause, all of this information I'm going to select and make categorical. It so happens that for this particular dataset, all of the columns have categorical values. The data types of these columns can remain unchanged. They're all numeric values; we just need to convert all of these columns to categorical columns. Having configured the Edit Metadata node, we can choose the Run selected option in order to run all of the selected nodes. I'm going to need more room to add in the other process in my ML workflow so I'm going to use the zoom-out option in order to make these nodes smaller so that I have more room to play around with. We'll perform hyperparameter tuning using our entire dataset. Head over to the Machine Learning option here on the left navigation pane. What we want is a classification model under Initialize Model. This time we'll choose a different classification model, one which has a number of model hyperparameters that we can tweak. The Two-Class Decision Forest algorithm is an ensemble learning technique, ensemble learning meaning it uses many different decision trees under the hood to arrive at the final result. Because it uses many different models, it's a more robust model overall. It brings the results of all these models together. Drag this machine learning algorithm that we want to use to build our classification model onto the center pane. Now we'll use the Properties panel to configure the hyperparameters for this model. We'll build this decision forest model using the bagging ensemble method. Now decision trees are high-variance algorithms, meaning they're very sensitive to changes in the input dataset. Decision trees are prone to overfitting on the training data and may not produce great models when used standalone. Bagging is an ensemble method to sample the data in the input dataset to build the different decision trees that make up the decision forest ensemble technique. Bagging is specifically used for high-variance techniques such as decision trees. It helps reduce the variance without worsening the bias. We can now use this panel to configure the the hyperparameters of this model. We'll have this decision forest have just one decision tree. The remaining hyperparameters we'll leave to the default values. The maximum depth of decision trees is 32, the number of random splits per node is 128, and the minimum number of samples that make up a leaf node is 1. Now that we've configured the algorithm, we can evaluate this particular model using cross validation. Under Machine Learning, Evaluate, choose Cross Validate Model, and drag this onto the center pane. Connect to its input the machine learning algorithm that we want to cross

validate. This is the Two-Class Decision Forest. Also connect the data that we want to use to perform cross validation, that is the output of the Edit Metadata. Bring up the properties of Cross Validate Model, launch the column selector to specify what this model should be trained to predict. We want to predict the class, whether the cancer was detected or not, based on all of the other features that we have. With the Cross Validate Model node selected, go ahead and click on Run selected in order to cross-validate this model and produce a validation score. Once cross validation is complete, right-click, go to Scored results, and Visualize. Along with the original features and labels at the very right, you'll find the Scored Labels. This is the predicted output from this model. You can generate the confusion matrix for trained model by comparing the scored labels with the class column, which is comprised of the actual labels or the actual values from the dataset. This is the confusion matrix showing you how many of the predictions from this model were accurate. Just looking at the numbers here makes it seems like the model was not a great one. Let's confirm this by closing out this dialog, right-clicking on the Cross Validate Model, and viewing Evaluation results by fold. Go to Visualize. In order to compare all of the decision tree models that we build, let's focus on only the accuracy metric. You can choose to use the precision or recall if you want to. There is one fold where the accuracy was above 0.46, but it tends to be around 0.65 if you view the rest of the folds. If you just select the Accuracy column, the statistics on the right will show you that the mean accuracy is about 0.59. That's not great for a binary classifier, and the median is around 0.64. Now that we have a baseline, we can compare this to other models that we build using hyperparameter tuning.

Tuning and Scoring Multiple Models

Since we are going to be training several different machine learning models for hyperparameter tuning, let's name these individual nodes so that we can distinguish them. Simply double-click on a node, in this case the Two-Class Decision Forest that we just trained, and this will bring up a little dialog where you can specify the name for that node. I'm going to call this node the Baseline. Once you've named the node, you'll see a little drop-down next to the node, which you can then expand and view the name. So expand this node and there it is, this is the Baseline model. Now let's configure other models so that we can compare this with the baseline. I'm going to copy this machine learning model over and create a new node. Right-click, copy, and then paste so we have one more copy of the Two-Class Decision Forest. This is the second model that we'll use for hyperparameter tuning. Open up the Properties panel and let's configure the hyperparameters. The only change I'm going to make here is to have the minimum number of samples per node be equal to 40. All of the other parameters I leave as is. One decision tree, max depth of the decision

tree is 32. I'm going to assign a name for this node right away, expand this, and switch the name over from Baseline to Minimum number of samples per leaf node 40. Remember that you need to double-click on this node to bring up this dialog. Once you've named your Two-Class Decision Forest, select the Cross Validate Model node that we used earlier, right-click, copy, and paste this node in so that we have a second copy of this node. In order to cross-validate this new model we need to connect the model itself to the Cross Validate Model node, and we also need to connect the data that we plan to use for cross validation. Let's go ahead and run cross validation on this new model that we have set up. Select the Cross Validate Model node and choose Run selected. Once the model has been cross-validated, we can right-click and view evaluation results by fold. Remember we are only going to take into account the accuracy of the model to see how well it performs, and based purely on the accuracy, you can see that this model has performed far better. Losing a minimum of 40 samples per leaf node, the mean accuracy is 0.64 and the median is 0.7. Now that we've got the hang of hyperparameter tuning, let's make another copy of the Two-Class Decision Forest baseline and paste it in to form a new node. With the new Two-Class Decision Forest node selected, open up the Properties panel and let's configure hyperparameters. This time I want to allow the maximum depth of the decision trees to be no more than 6. Limiting the depth of the decision tree will prevent overfitting on the training data. Now that we have the model configured, copy over the node Cross Validate Model, right-click, copy, and then paste in, and connect the new model that we set up, the Two-Class Decision Forest to this Cross Validate Model node. The other input that you need to specify to Cross Validate Model is the data. Connect the output of Edit Metadata to Cross Validate Model, and I'm now going to double-click and rename this Two-Class Decision Forest node so that it says Maximum depth of the decision tree is 6. Select the Cross Validate Model and go and Run selected nodes. Once the cross-validation process is complete, select the Cross Validate Model node, right-click, go to Evaluation results by fold, and Visualize. Select the Accuracy column, and you can see that the mean accuracy of this model is 0.66 ; the median is far better, 0.71. I'm going to set up one more model with different hyperparameters. Copy the Two-Class Decision Forest baseline, and paste it in to form a new node. Select this new model and bring up the Properties panel so that we can configure hyperparameters. This time I'm going to have the number of random splits per node be equal to 50. Let's identify this node by giving it a meaningful name. I'm going to double-click and rename this to Number of random splits 50. Now the steps are exactly the same. Copy over the Cross Validate Model node, paste it in, connect the machine learning model to this node, and the second input to Cross Validate Model is the data used to train the model. The process is familiar to us. Select the Cross Validate Model node, run selected, wait for the training process to complete, and once that's done, right-click on Cross Validate Model, go to Evaluation results by

fold, and Visualize, and when you check the accuracy of this model, you can see that the average and median values of accuracy have fallen a bit, 0.58 and 0.63. The model is significantly worse than the previous one.

Summary and Further Study

And with this, we come to the very end of this module on hyperparameter tuning. We started this module off by understanding what exactly hyperparameters are and what it means to perform hyperparameter tuning on your model. We discussed in detail the difference between model parameters that are found during the training phase of your model versus model hyperparameters that remain constant during training and are part of your model's design. We looked at a quick overview of how decision tree classifier models work, and we performed hyperparameter tuning using Azure ML Studio on a decision tree classifier. And with this, we come to the very end of this course on Experimental Design for Data Analysis. If you're interested in data and want to work on the Azure Cloud Platform, here are some other courses on Pluralsight that you might find interesting: Summarizing Data and Deducing Probabilities will talk about uni-variant, bi-variant, and multi-variant analysis of data; and Communicating Data Insights will talk about the different visualization techniques that you can use to express insights. And that's it from me here today. Thank you for listening. I hope you enjoyed this course.

Course author

Janani Ravi

Janani has a Masters degree from Stanford and worked for 7+ years at Google. She was one of the original engineers on Google Docs and holds 4 patents for its real-time collaborative editing...

Course info

Level	Intermediate
Rating	★★★★★
My rating	★★★★★

Duration 2h 44m

Released 20 Jun 2019

Share course