How to Think About Machine Learning Algorithms

by Swetha Kolalapudi

Course Overview

Course Overview

Hi everyone, my name is Swetha Kolalapudi, and I'd like to welcome you to my course, How to Think About Machine Learning Algorithms. I am the co-founder of a start-up called Loonycorn. Machine learning is all the rage these days, but too many folks get intimidated by its reputation. Contrary to popular perception, you don't actually need to be a math genius to successfully apply machine learning. Machine learning techniques can be learned from first principles by anyone who has the will to learn. This course is all about learning machine learning from first principles. There's no jargon, no abstruse math, just simple direct explanations and techniques that you can use directly. By the time you are done, you will know how to set up trading stocks, recommending movies to friends, or sensing the sentiment about your favorite candidates, as cookie-cutter ML problems that you can write, code, or solve. Some of the major topics that we will cover include, classifying data into pre-defined categories, predicting relationships between variables with regression, recommending products to a user, and clustering large data sets into meaningful groups. By the end of this course, you will be able to recognize opportunities where you can use machine learning and solve problems using standard techniques such as support vector machines, or linear regression. Before beginning this course, you should be familiar with Python at

a very basic level. I hope you'll join me on this journey to learn how to think about machine learning algorithms at Pluralsight.

Introducing Machine Learning

Recognizing Machine Learning Applications

Hi, I'm Swetha and I'd like to welcome you to this introductory course on Machine Learning and how to really think about and approach Machine Learning problems. Through this first module, I'd like to help you double up an intuitive feel for what Machine Learning is and where it can be used. At the end of this module, you should be able to spot applications of Machine Learning around you in the real world. You should be able to differentiate between the different types of Machine Learning problems that exist. And you should be able to pick which type of problem it is, what you are trying to solve. It turns out that Machine Learning is actually the driving force behind a lot of applications, services, and products that we have just come to take for granted. Think brands like Amazon or Netflix, which are known for their amazing service. And a big part of that service is how they seem to be able to personalize their service and experience for every user in a way that perfectly matches their tastes and needs. Sometimes these services seem to understand what you're looking for before you even know it yourself. Then there are amazing products, like Siri or Google Maps acting like indispensable personal assistants catering to your every need and making your daily routine much easier to manage. Even those tiny little details like an application that can auto complete a sentence or spell correct your mistake in spellings or an email service that knows what is important to you and what is spam. And then there are those innovations that are really ushering in the future. Whether they're self-driving cars or the internet of things. So what's the common theme among all of these applications and products and services? You'll notice that as I was talking about each of them, I was referring to them as if they have some sort of intelligence of their own. And behind the scenes of all of these applications, it is Machine Learning that is helping them derive this seeming intelligence. So now that you know that Machine Learning is behind so many of the ordinary and very cool things that we see around us, let's take an example to help us understand what exactly is going on in Machine Learning. Say you're an alien observing Earth from far away and you're looking at one particular park on Earth. You happen to know somehow that there are two types of human beings on Earth, males and

females. And you want to classify visitors that you see coming to the park as male or female. You don't really know what are the distinguishing characteristics that help you differentiate between male and female humans, but let's say you can see that folks who are taller than six feet seem to be very different from folks who are shorter. So you just decide to say that if the height is greater than six feet, you'll assign one gender, say males, and call all the rest females. This is a rule based approach and basically what you've done is apply some sort of logic to identify a rule or rules that help you perform the task you've been assigned. Here's a different approach. Let's say you somehow kidnap a human shopper and you have the human pointing out the gender of each visitor as they visit the park. After observing how a few hundred visitors have been classified, you intuitively learn how to differentiate between the genders. This is a Machine Learning based approach. There's something a little bit familiar about this approach. You would have realized that the Machine Learning based approach is quite similar to how humans learn. Imagine how you learned things when you were a child or how you learn things at work now. Human beings learn to identify patterns. When they're exposed to some phenomena for a very long period of time, that's how you learn to speak, that's how you learn to do your work. Human beings are able to learn how to perform tasks based on experience. Machine Learning is the process by which a computer program or a system is able to learn how to perform a task based on experience. In the case of a computer program or system, experience is actually data. Just like the alien collected data from the kidnapped human about which visitors were male and which visitors were female, a lot of systems based on Machine Learning capture data. And as more and more data is being captured in more and more applications these days, Machine Learning is becoming ubiquitous. For instance, if you look at User Clicks and Views data, it's being used to build Recommendation Systems which enable to identify what a user might want or need even before a user themselves. A voice assistant like Siri basically learns from questions and answers collected from users in the past. The more users use Siri, the more it learns and the better it becomes at answering questions itself. So the next time you come across a problem where you need to teach a computer or a program how to perform a task and you've got a lot of data, ask yourself whether you should use Machine Learning to solve that problem.

Knowing When to Use Machine Learning

Let's take a service like Google Maps, which many folks use on a daily basis. The cool thing about Google Maps is that it not only tells you directions to get from point A to point B, it also tells you how long it will take you to travel from point A to point B at that current time. Let's say you work for a maps company and you want to write a similar program. There are two approaches that you

could take here, as we've seen in a previous example. A rule based approach or a Machine Learning based approach. Machine Learning might seem very cool but it shouldn't be a default choice to solve a problem. So let's take an example that we just set up and understand what are the characteristics of a problem that really calls for the use of Machine Learning. If you took the rule based road, then you would have to define a large set of rules that help you calculate the commute time. This large set of rules has to be defined by you or the human programmer who is setting up the system. A rule might look like this. If the day of the week is Monday, and the time of the day is between 10 AM to 11 AM and the distance to be traveled is two kilometers to four kilometers, then the commute time should be 40 minutes. This is just one rule. You'd have to come up with a whole bunch of such rules, maybe thousands. And these rules are identified manually by human analysts after a lot of research. Maybe you had a bunch of people driving around at different times of the day on different days of the week, collected all their learnings, summarized it in some sort of table, and then use that as your rules. This is actually a tried and tested way of doing things. Whether it's financial analysts deciding whether to invest in some company or not, or whether it is a marketing professional who is deciding what kind of discounts they want to give. They base their decisions on rules that they have identified through their research. Now rules that are identified like this are actually static. They don't really change very often, and if they change they need to be changed by the human analyst based on new research that they might have identified. There's a problem with using static rules for the commute time calculation problem, though. And that is that traffic patterns are actually dynamic. There are a lot of different variables which influence the traffic patterns. It could be weather, the number of visitors to town, whether it's a holiday or not. And as these things keep changing, the traffic responds differently. So just coming up with a static set of rules based on a simple set of variables that using and learns the traffic may not be good enough. The other option is to use the Machine Learning based approach. Now in the Machine Learning based approach, there'll be three steps. First you would collect a large amount of traffic data. There might be tens or hundreds of different variables that you think influence the traffic data and you need to collect all of that and keep collecting it on a real-time basis. That's actually what the Google satellite does. Then you would use an algorithm that would identify by itself what are the relationships within those variables and the commute time and you would update this relationship continuously as you get more and more data. If we had to really pick out what's the key difference between rule based and Machine Learning based approaches, in the rule based approach, you would apply a set of static rules on the current context and derive the commute time whereas in the Machine Learning based approach you would again apply a set of rules but the difference is that those rules are updated automatically based on data. Let's take a look at how this would actually work. Here's

the system where the input is the current context and the output is commute time, but the difference here is that the rules get updated based on historical data and that's what's actually called for in this particular situation. So here's when you should use Machine Learning. Use it when it is really difficult for humans to express the rules involved in performing a task because of the number of influencing variables involved and because of the complex, difficult to understand relationships between those variables. Another important prerequisite or requirement to use Machine Learning is that you definitely need to have a large amount of historical data available and finally if the rules or the patterns and relationships among the data are dynamic and keep changing with time then Machine Learning is perfect for that use case.

Understanding the Machine Learning Process

At this point, you should be able to look at the problem and decide whether you want to take the Machine Learning route or not. If you do decide to take the Machine Learning route, there is actually a pretty structured process that you need to follow. Identify the type of Machine Learning problem that you're trying to solve. Represent the data that you have using numeric attributes. And finally, apply a standard algorithm on that data. Follow these three steps and you should be able to solve any Machine Learning problem. Let's walk through each of these steps. Most Machine Learning problems can actually be expressed as one of these four different types. Examples of which are sentiment analysis and spam detection. Regression problems, examples of which are problems like sales forecasting or predicting the value of a stock market index. Clustering, customers' recommendation is an example of clustering. Or recommendations, such as product recommendations that an e-commerce site might want to give to its users. It's really crucial that you pick the type of problem right, because each type of problem has its own type of workflow. How you set up the problem statement and how you represent your data depends upon the type of Machine Learning problem that you decided to solve. Now as you know, if you have decided to use Machine Learning you definitely need to have a large amount of historical data that you're going to use as input to your algorithm. Data might be available in the form of unstructured text, or images, or videos. But we use this data in a Machine Learning exercise, it's important to use meaningful numeric attributes to represent the data. For example, you might take an image and represent it using numeric attributes like the height and width of the image and a color intensity value for each pixel in the image. Choosing meaningful attributes that are relevant to the problem you're solving is crucially important. There's no point feeding in the data of how many caterpillars were born in India to a program that is going to predict the weather in California tomorrow unless you think there's some sort of weird butterfly effect involved. Finally,

you would take the data that you have represented using numeric attributes and feed it to an algorithm. The algorithm's job is to use the data that you feed it as a learning experience and identify patterns which it will represent in the form of rules. The rules are nothing but a quantitative representation of the relationships between variables. These rules that the algorithm has found together become a model. In a spam detection problem, the model might represent the relationship between the text in the email and whether it is a spam email or not. The model is usually represented using some sort of mathematical equation or a set of rules which might be if-then-else statements. The main thing the user needs to do in this step is to choose an algorithm that's going to find the model. The choice of algorithm depends on the type of problem that you're trying to solve. If you're trying to solve a classification problem, then you could choose between algorithms like the Naive Bayes algorithm, the Support Vector Machines algorithm, or many other different algorithms which are available. If you are trying to solve a clustering problem, you could choose between algorithms like K-Means or Hierarchical Clustering. The thing is though, choosing the algorithm is not really the most important part of a Machine Learning exercise. You can usually plug and play different algorithms with minimal effort, because a lot of different programming languages have pre-built packages for algorithms. So the choice of algorithm could be a model of experimentation rather than knowing beforehand which algorithm you actually need to use. On the other hand, setting up the problem right and representing the data correctly is crucial and a lot of thoughtful choices, wisdom, and experience goes into getting these right.

Identifying the Type of a Machine Learning Problem

We've looked at what are the three steps involved in a typical Machine Learning exercise. One of the most crucial steps is identifying what type of problem that it is we're trying to solve. This especially depends on how you set up the problem statement. You might take a qualitative problem and express it using word classification and regression. Setting up the problem statement and choosing the input and output representation for your task involves a lot of thoughtful choices. What we'll try to do now is to go through each of the different types of ML problems, Classification, Regression, Clustering, and Recommendations. And see what types of problem statements could be defined under each of these types of Machine Learning problems. At the end of this, we're given a problem statement, you should be able to see whether it's a classification, regression, clustering, or recommendations problem. Here are a few examples of Machine Learning applications which are classification problems. Spam detection in your inbox, sentiment analysis, and identifying a trading strategy, whether to go long or short on a stock. The

problem statement is actually a question. For example, in spam detection, the question might be, given an email, is this email spam or ham? Ham is what non-spam emails are called. In Sentiment Analysis, the question might be, given a text, like a tweet, is the tweet expressing a positive or a negative sentiment? Let's say you wanted to identify a trading strategy, whether you want to go long or short on a given date. The question you might ask could be, is this trading day going to be an up-day or a down-day for this market? All of these problem statements have certain common characteristics. They involve taking an object or an entity and classifying it. The object or entity to be classified is called a problem instance. In spam detection, we were trying to classify an e-mail. In sentiment analysis, we were trying to classify a Tweet. In the creating strategy example, we were trying to classify a trading day. So on the one hand you have something that needs to be classified, on the other hand you have categories into which it can be classified. There could be two categories, in which case we are doing final reclassification or there could be many categories and we need to choose one of those categories. There's an important prerequisite if you decide to go with the classification route. Classifiers, which are algorithms that can perform classification required a set of instances for which the correct category membership is already known. This set of instances is called the Training Data. For instance, in sentiment analysis, you need a set of Tweets that has already been classified as positive or negative and the classifier will learn from that how to classify new Tweets into the right categories. Let's move on to Regression. Here are a few different examples of problem statements which correspond to regression. A quantitating problem set of regression could read at, what will be the price of a stock on a given date? An application like Google Maps needs to answer how long it will take to commute from a point A to a point B. A marketing professional who's planning promotions might want to forecast what the sales of a given product might be in a future week. The important characteristic of a regression problem is that we are computing a continuous value as compared to classification where the output is categorical. All the outputs here in the problems we saw, the stock price, commute time, the sales forecast, are all continuous values. They could take any value within a range of numbers. So that's the output that you're looking for, some continuous value, and you know that or can hypothesize that, that value, depends upon certain influencing variables. For instance, the commute time, you could hypothesize the depends upon the time of the day and the distance between point A and point B. If you had to express this visually, you could see that we are taking input variables like the time of day and the distance, we're applying some function, this could be some mathematical equation on top of these variables, and as an output we get some number which corresponds to the commute time. The job of the regression algorithm is to identify that function that relates the commute time to the time of day and distance. Like classification, regression also requires training data. This is basically a large set of historical data

points with the values for both the input variables and the output variable. The regression algorithm uses this training data to identify that function that relates the input and output. Moving on to clustering, here is an example where you might use clustering. Say you have a large group of users for a Social Network. You want to divide the users into groups based on some common attributes so that within a group the users are similar to each other. The key thing here is that the groups to be divided into are unknown beforehand. So you're not starting out like in classification with a set of groups that you know that the data belongs to. You're using the algorithm to identify what groups might exist. Take the entire list of users of the social network and let the algorithm divide the users into some groups. Later on you might realize that these groups actually represent meaningful divisions, for example, demographic based division or divisions based on likes and dislikes of the user. When might you use clustering? Let's say you wanted to do a user segmentation of all the users of Facebook. You would use clustering to identify what meaningful groups of users exist. Then whenever a new user comes in, you would actually use a classification algorithm to assign that user to one of these existing groups. Lastly, let's talk about Recommendations. Recommendations are something you see in most online services and products these days, from radio stations that are able to answer what kind of artists a user might like or e-commerce websites that are able to tell a user what are the top 10 book picks for that user or what else they might need to buy if they buy a phone. The basic key here is to take a user's past behavior and from it, determine what kind of things they might like or need further. In the Machine Learning community, another theme that this type of problem carries is Collaborative Filtering. That brings us to the end of all the different types of Machine Learning problems. Given any problem, you can set up your problem statement for one of these different types of problems and the choice that you make here in how you set up that problem statement will completely determine what happens when the next steps of the Machine Learning exercise. That brings us to the end of this module, introducing Machine Learning. By now you should be able to look around you and spot applications of Machine Learning around you, whether it's in your email inbox or whether it's in your business. You know the different types of Machine Learning problems which are available and you should be able to pick the type of problem that you're solving, based on your problem statement. In the next module, we'll be looking at several different examples of classification problems and how we can set these up in the Machine Learning workflow.

Classifying Data into Predefined Categories

Understanding the Setup of a Classification Problem

Welcome back to this introductory course on machine learning. In this module we're going to look at classification problems which involve classifying data into a set of predefined categories. We'll be learning how to recognize and spot classification problems around us, and given a situation where you think classification is appropriate, you'll be able to set up the different elements of the problem. The problem statement, the inputs to classification, which are called features, and the output of classification which is called a label. Given an e-mail, could we classify it as a spam or a ham e-mail? That is what non-spam e-mails are called. Given a tweet, could we classify it as expressing a positive or a negative sentiment? Given a trading day, could we classify it as being an up-day where the prices of stocks go up, or a down-day, where the prices of stock go down? All of these are examples of problem statements for classification problems. All classification problems usually have the same setup, the same set of steps that you need to follow. First, you need to define a problem statement. Classification problems usually have a typical form in which they're expressed. Now as you know, machine learning problems involve data. You take an algorithm and you feed it data and the algorithm learns patterns from the data. This data needs to be represented using numerical attributes called features. Then you have a phase where the classification algorithm learns from a set of data called the training data, and finally you apply the learnings or the patterns that you got in the training phase to help you classify new data. Let's get a little bit more detail on each of these phases. First we have to set up a problem statement. This has a typical form. We take a problem instance, this could be, for example, an e-mail or a tweet or a trading day. These were the problem instances in the questions that we saw earlier representing classification problems. In a sense, a problem instance is an entity or an object that we want to classify. To solve a classification problem, the problem instance is basically fed to a classifier which can assign a label. The label is the category which is being assigned to this problem instance. An e-mail has to be classified as Spam or Ham. Spam and Ham are the labels here. A tweet has to be classified as positive or negative. Positive and negative are the labels. In the trading example, up-day and down-day are the labels. This is the typical form of any classification problem statement. You have a problem instance and you want to assign a label to that problem instance from a given set of labels. Once again, problem instance is just a name for what we want to classify and labels is the name for the set of categories that we

want to classify it into. Think of the classifier as a black box. It just takes a problem instance and assigns a label. This is exactly the machine learning objective: to build this black box classifier. What happens inside this black box is represented using some sort of mathematical rules or equations, and it is called a model. The second phase in classification is to represent your data using numerical attributes. This is standard in any machine learning workflow. Classifiers are basically mathematical or statistical algorithms, and any data that you feed to this algorithm needs to be represented using numerical attributes that are meaningful. So even if, for example, you're classifying text or images, you need to represent them using numerical attributes before you feed them to a classification algorithm. The next phase in classification is called training. What happens here is that you take a large amount of historical data, this is a set of problem instances which are already correctly labeled. For instance, you might already have e-mails which are explicitly marked by users as Spam or Ham in their inboxes. This set of correctly labeled instances is called training data. In the training phase, the classification algorithm basically learns from the training data. It tries to identify rules and patterns based on the training data. Each data point in the training data is a tuple of features and a label. In the e-mail example, you have a large set of e-mails. Each e-mail is represented using a tuple of the numerical attributes representing that e-mail, and whether it is a spam or a ham e-mail. The patterns that the classification algorithm learns from the training data represented in this way constitute the model, or the black box that we wanted to build that could classify any new data. Not all machine learning techniques follow this same workflow, where you have a set of training data for which the output required is already known. If you do have such a set of data, then you can have a training-the-model phase, and the technique that you're using comes under a set of techniques called supervised learning techniques. Finally we come to the test phase where we're actually classifying new data that we haven't seen before. So we've built a classifier, this is a black box that is capable of classifying any problem instance and assigning a label to it. So a spam-detection classifier can take an e-mail and assign a label of Spam or Ham. The problem instances that we feed to the classifier in the test phase are instances that the classifier has not seen before. The efficacy of our machine learning solution, the classifier that we built in the training phase, is determined by how well this classifier can correctly classify instances that it has not seen before. So those are the four steps involved in a typical classification problem. Setting up the problem statement and using the right features to represent your data is crucial, and this is where a user or a machine learning expert would spend a lot of time putting in a lot of careful thought. In the rest of this module, we'll be looking at how to set these two up for a bunch of different examples. Once you've got the problem statement and the features to represent the data, you would just take your data and feed it to a standard algorithm, and most of these algorithms are available as pre-built libraries for platforms like

Python, R or Spark. You would have a method for training that algorithm and for testing, and you can just plug and play the algorithm once you have your data set up correctly. Naive Bayes, support vector machines, decision trees, K-nearest neighbors, random forests, logistic regression. These are all examples of algorithms that you can plug and play to solve classification problems once you have the problem statement and features set up.

Detecting the Gender of a User

If you worked for or ever used any online service, then you're very familiar with a registration form that looks like this. Most online services use this registration form to collect customer information. The reason they do this is that if they know demographic information like the gender of the customer, that can help the business create targeted offers and personalized experiences for specific customers based on their demographics or other characteristics that define them. The problem, though, is that most folks will only fill in fields that are marked mandatory. Usually, the name, the screen name, the date of birth, these are fields that might be mandatory for the service to be able to register you as a customer. And in general, fields such as gender will not be made mandatory, and the result is that most folks will not fill it out. Only a fraction of the users will fill out this field. In general, it's not preferred that we make this a mandatory field to fill, because the more fields that the user has to fill, the less likely it is that they'll actually complete the registration process. So you would like to keep only as few fields mandatory as necessary. So how does this service, which wants to know what the gender of their user is so that they can create good offers, find out the gender of the users who did not fill in this particular field? Since the name of the user has been filled in for every user, given the first name of a user, can the system make a good guess to whether this user is male or female? This can actually be set up as a machine learning classification problem. Let's see how to do that, looking at the four different phases of a machine learning classification setup. What we'll do is to define a problem statement and the features for the gender-detection scenario. As we know, the problem statement in classification has a particular form. We need to take some objects and classify it into a set of categories. In this case, we have the first name of every user. We can take the first name and assign a label, male or female. So this perfectly fits into the classification problem statement form. Now all we need to do is to take every name and represent it using numeric attributes, which we're going to call features. Now to do this, we can use characteristics that usually differentiate male or female names. For instance, in many cultures, female names end with a vowel. So you could have a binary number, one or zero, to represent whether the last letter of the name is a vowel or not. Another example of a feature is the number of characters. It could be that the number of characters

differentiates between whether the name belongs to a male or a female. There might be some prefixes or suffixes common to a specific gender, and you can use binary attributes, ones or zeroes, to represent the presence of those prefixes or suffixes. As you can see, a lot of thought needs to go into what kind of features you can use to represent the name. Getting these features right does sometimes involve trial and error, where you might try out different features and look at the efficacy of the algorithm with different sets of features. Once you have the problem statements and the features set up you can move on to the training phase. In the training phase, you would take all the names of folks who did fill in their gender and feed that through a standard algorithm. Once the algorithm has trained a model, you would then feed new names to the model to figure out whether they are male or female names.

Classifying Text on the Basis of Sentiment

Consider a product launch, how do you know whether it has gone well or not? Consider that you are a candidate in an election, how do you figure out what voters are feeling towards you? If you ran a brand, what do customers think or feel about that brand? All of these questions involve analyzing how people feel about something. And how people feel about something, whether they are feeling positive, happy, or whether they are feeling negative, that is angry or unhappy, can be measured using a technique known as sentiment analysis. The idea of sentiment analysis is not new, but the reason it has become very relevant and exciting today is because of the availability of opinions of folks on online forums. These opinions are generally in the form of tweets of reviews or comments, and this data is huge. There might be hundreds of thousands of tweets, reviews and comments. This data has some important characteristics. It's unstructured text, which is semantically complicated to parse to extract any information from it, but it is freely and publicly available for anyone to analyze. The data represents a treasure trove of insights that we can draw from it, and we can use sentiment analysis to draw those insights. Basically, what sentiment analysis is trying to do is to take a bunch of comments and say that some of these comments are expressing negative sentiments, and some of them are expressing positive sentiments. Out of all the comments about the particular topic, if we knew how many were positive and how many were negative then we could compute a general positivity or negativity score about that particular topic. So the key challenge here is, given any comment, we need to be able to determine whether it is positive or negative, and this is called identifying the polarity of a comment. Just by looking at this problem statement, identifying the polarity, whether a statement is positive or negative, this is a classic example of a classification problem. Now we know that there are four steps involved in any classification setup. Let's look at how to set up the problem

statement and the features for a sentiment analysis scenario. For the problem statement, we need to put it in the typical form, which is classifying a problem instance into a set of labels. Here the problem instance is a comment that we want to classify, and the labels are positive or negative sentiment. The next step is to use features to represent the data. This is a little bit complicated because we have unstructured text here that we need to represent using numeric attributes. So, what kind of numeric attributes can we use to correctly represent the text? One meaningful way to represent text using numeric attributes could be using the words present in that text. You could start by creating a list representing the universe of all words that could appear in any text or any comment in the world. So let's say this list, W1 to WN, represents the complete list of all words that can appear. For the sake of clarity, let's just assume that W1 is the word hello, W2 is the word this, and so on, WN is the word goodbye. Any text can then be represented using the frequencies of these words. Let's see an example. Here is a sentence: hello, this is a test. We'll represent this sentence using a tuple of N numbers because N is the total number of words present in the universe in our scenario. Here is the tuple representing this sentence. Each number in this tuple represents a count of a particular word in this sentence. So the first number represents that hello occurs once in this sentence, the second number one represents that this occurs once in this sentence, and so on. If a word does not occur in the sentence, then the frequency is zero. This sort of representation where you represented a sentence using the frequencies of words in that sentence is called a term frequency representation, and it's a very popular way for representing text to be fed to machine learning algorithms. Now that we know what features to use to represent a comment you can move on to the training phase. In the training phase you will take a comment's dataset where you have comments already labeled as positive or negative. You would represent each comment using the term frequency representation and feed the data to a standard classification algorithm and go through the training and test phases.

Deciding a Trading Strategy

Let's consider a quant trading scenario. Say you work for a hedge fund and your job is to trade stocks on a stock exchange. So you need to decide whether to buy a stock or to sell a stock at any given point in time. You would buy the stock if you believe that the stock's price is going to go up in the future, so that when you sell it you can make a profit. You would sell a stock if you believe that the stock's price will go down. So on any given trading day, on that day you need to decide whether you want to buy the stock on that day or do you want to sell it. Buy it if you believe that during the day the stock's price is going to go up, and sell it if you believe that the

stock's price is going to go down. This can be set up as a classification problem. When you're told to build a quant trading model, classification is not something that you would think of immediately, so this is an interesting use case for classification. As usual, we need to ask: what is it that we want to classify? And in this case we want to classify a particular trading day. We want to classify it as an up day for a stock, where the stock price is going to go up, or a down day for the stock, where the stock price is going to go down. So we've taking a quant trading problem and expressed it as a classification problem statement. Let's look at what the features should be in this case. So we need to take a trading day and represent it using some numeric attributes that are indicative of whether the stock price on that day is going to go up or down. This is where the trader will bring in his or her domain expertise. For instance, one of the features that they might choose to represent a trading day is the day of the week, and similarly, the month of the year. These are calendar features which will capture any seasonality effects or trends that are involved in influencing a stock's price. For instance, quarter-ending dates where annual reports of companies are released might be dates which affect the price of a stock and the direction that it will move in. You could look at the price of the stock on previous days, or the price of related stocks on previous days. Related stocks could be stocks in the same industry. For instance, if you're looking at Google's stock, then you could look at other tech stocks like Apple or Microsoft, and what has been happening with their prices in the previous days. There's a lot of room for experimentation here in which stocks you might use, how many days of history you might include for one trading day; all of this is subject to the trader's knowledge and experience, and how well the model is performing with each scenario. Now you would move on to the training phase. In this phase you would collect financial data for the last 10 years or so, and represent each trading day in the last 10 years as an up day or a down day for the stock that you're interested in. Now each trading day is a tuple of its features and whether it's an up day or a down day. Feed this to any standard classification algorithm and go through the training and test phases.

Detecting Ads

A lot of websites use ads, and there are users who are always looking for extensions which will help them block ads, to make for a better experience. Let's say you want to build an ad-block extension for a browser. What the browser will do is, when it is rendering images, it needs to identify any ad images and then block them out. So given an image, how will the extension know whether it is an ad that has to be blocked or not? This can be set up as a machine learning classification problem. So let's look at the four steps that we already know are important for a classification setup, and see how to first set up the problem statement and features for an ad

detection scenario. To set up the problem statement, we need to know what we want to classify, in this case that is an image that is being rendered by the browser, and we want to be able to classify it as an ad or not an ad. So the labels are Ad and NonAd. So setting up the problem statement was pretty straightforward but what we need to do is to take images and represent them using numeric attributes. Now the height and width of an image are immediately numeric attributes that jump to mind, but we also need other attributes which are more indicative of whether this is an ad or not. This could be the URL of the page where the image is being rendered, the URL of the image itself, the text on the page, and the image caption text. Comparing all of these would give us some idea as to whether this is actually an image that is relevant to that page, or is whether it is irrelevant and is actually an ad image. Now we said we wanted numeric attributes, but the URLs and the page text and caption text are actually text attributes. We've already seen in the sentiment analysis example how we can take text attributes and represent them as numbers, using methods like the term frequency representation. So you can take these text attributes and represent them using some numeric representation. Now you can move onto the training phase. In this phase, you basically need an image dataset where all the images are already labeled as Ad or NonAd images, and each image has been represented using these attributes that we identified. Feed this data to any standard algorithm and run the training and test phases and you'll have a machine learning setup where you can classify any image as Ad or NonAd.

Understanding Customer Behavior

There are several instances in which machine learning classification can help us understand customer behavior. Businesses often study customer behavior in order to draw insights that are helpful for the business. One such problem is customer churn, where you want to study a customer's behavior to understand if they are going to stop using your service in the future, and if they are, you would like to take some preemptive measure. Fraud detection is another example where you would study customer behavior to figure out if that customer is going to commit some sort of payment fraud. Credit risk is another example where you would like to understand whether a customer is likely to default on their payment or loan based on their current behavior. All of these are examples where we're using current customer behavior to predict whether some event is going to happen in the future: whether the customer is going to stop using the service, whether they're going to commit fraud, or default on some payment. However, each of these can actually be set up as a classification problem, where you are classifying something in those set of categories. Let's quickly run through each of these problems and see how it can be set up as

classification. First we have customer churn. The problem instance, what we want to classify here is a customer. What do we classify them into? Two sets, a set of customers who will repurchase, and a set of customers who will not repurchase from our business. How do we represent this customer? Using purchase behavior in the recent past, their demographics and the days since their last purchase. So if it's been a pretty long time since they've purchased with us, it's likely that the customer's not going to repurchase again. For training data, you would take a snapshot of customer behavior in the past, and then you would say, at that point, did the customers then go on to purchase or repurchase? That would become your training data. All you need to do now is to feed this to a standard classification algorithm. Let's move on to the next example, which is fraud detection. What we want to do here is take one particular payment and then say whether it will turn out to be fraud or not fraud. So that is a problem statement. We take a payment and classify it as fraud or not fraud. The features that we might use are the payment type, the frequency of use of that particular card, the number of failed attempts for payment that this customer has made in the last hour. You would take a large number of historical transactions which are already marked as fraud or not fraud, represent them using these features, and feed it to a classification algorithm. Moving on, let's talk about credit risk. The problem statement here is to take one customer and be able to say whether they are going to default payment or they will not default payment. So here we have a classification problem statement. The relevant attributes you could use to represent a customer here are the income of the customer, the education level, their employment sector, and their history of whether they have defaulted or not. You would have some sort of score for each of these attributes. For training data, you would take historical data of customers labeled as defaulted or did not default, and you would represent them using their particular attributes at that point in time, in terms of income, history of defaults and so on, and feed it to a classification algorithm. So that brings us to the end of this module. Let's quickly summarize what we've learnt in this module. We've gone through a lot of different examples of classification problems in different fields, from spam detection to sentiment analysis to quant trading to image ad detection. We've seen how to take a business problem in these domains and represent them as a machine learning classification problem, including setting up the problem statement, the features and the labels for that particular problem. Now that we've become comfortable with recognizing classification problems in the world around us, we'll look at specific classification algorithms and how to use them to solve the problem. That is, how do we go through the training and test phases with specific classification algorithms, like the Naive Bayes algorithm or the support vector machines algorithm.

Solving Classification Problems

Using the Naive Bayes Algorithm for Sentiment Analysis

Welcome to this module on solving classification problems. We've already learned how to recognize and identify classification problems in the world around us, and how to set up the problem statement and features for several different problems. The next step would be to actually solve that problem by feeding the data through a standard algorithm. And that is what we will be learning how to do in this module. In this module, we'll concentrate on two specific classification algorithms, the Naive Bayes and the support vector machines algorithm. We'll understand how the Naive Bayes algorithm works, and then implement it to solve the sentiment analysis problem. Then we'll understand how the support vector machines algorithm works, and use it to solve the ad detection problem. At the end of this, you should be comfortable using either of these algorithms to solve machine learning classification problems. So let's go back to the sentiment analysis problem. We had already talked about setting up a problem statement for this problem. The problem statement is, to take a comment, and classify it as positive or negative. In this case, the problem instance is a comment, and a label is either positive or negative. That was the first step in the classification set up. Now let's talk about how to set up features for comments, that will be fed to a machine learning classification algorithm. In the sentiment analysis case, to solve this problem, we need training data, which is a large body of texts or comments which are already labelled as positive or negative. Each of these texts, needs to be represented using numeric attributes or features. We've already talked about how we can represent text, using the frequency of words which are present in the text. This is called a term frequency representation. Each text would basically then be represented by a tuple of numbers, which represent the frequencies of the same words. The next step would be to feed this data to a standard machine learning algorithm. Let's talk about the Naive Bayes algorithm, and what happens in the training phase of the Naive Bayes algorithm. The Naive Bayes algorithm is actually really intuitive to understand. In the training phase, the algorithm is trying to collect information that is going to help it to classify any new text as positive or negative. One piece of information that might help with this, is, out of all the texts which are present in the training data, how many are positive and how many are negative? So, P naught, is the percentage of all the texts in the training data, which are positive, and one minus P naught is the percentage of all the texts which are negative. This information is useful because if you pick a comment at random it will tell you

what the probability of that comment being positive or negative is. For the sake of clarity let's just assume that we were able to compute these numbers from a body of text, and these percentages turned out to be 55% and 45% respectively. Now, comments are actually made up of words. So each element in those tuples there represents the frequency of some word in that comment. These frequencies can be used to compute a positivity score and a negativity score for every word, that might be encountered in any comment. For instance, you can take the word 'happy, ' and compute a _____pos score for the word 'happy' as the sum of frequency of the word 'happy' in positive comments, divided by the sum of frequencies of the word 'happy' in the entire corpus, that is in the entire training data set. This is a measure that tells you, out of all the times that the word 'happy' might occur, how many times does it occur in positive comments? Similarly we can calculate a neg score for the word 'happy, ' out of all the times that it occurs in the entire training data set, how many times does it occur in negative comments? This would simply be, one minus the pos score of 'happy. ' We'll do this for every word that might occur inside the training data. And at the end of it, you'll have these two pieces of information. The probability that any comment picked at random is a positive comment, or a negative comment, and the pos score and neg score for every word present in the training data. Let's now see what happens in the test phase of the Naive Bayes algorithm. Given any new comment or piece of text, like for example the sentence, love the food, the algorithm should be able to calculate a positivity score and negativity score. It does this using the pieces of information that we got in the training phase. All we need to do is to assign the label based on which of these scores is higher. To compute the pos score of a comment, use the pos scores of the individual words in that comment. So you have the comment, love the food. To compute the positivity score, multiply pos score of love, pos score of food, and the overall probability that the comment is positive. In general, in cases like this, we ignore words like 'the', 'a', 'and' and so on, which are false stop words and don't really add any real information and don't show the sentiment of a given comment. If we use the numbers that we assume to have been computed in the training phase, then this score for the sentence will turn out to be, 0. 47. Similarly, you can go ahead and calculate the negativity score of this sentence. You can use one minus the pos score of the word 'love, ' one minus the pos score of the word, 'food, ' and one minus the overall probability that the comment is positive. We can substitute these values using the numbers that were computed in the training phase, and the negativity score at the end of this calculation will turn out to be 0. 01. So now that we have both the positivity and negativity scores for the sentence, love the food, if we compare the two, it is clear that the sentence is positive. In general, this is the form that the pos score will take. We multiply the pos scores of the individual words which are present in the comment, and finally multiply it with the overall probability that a given comment might be positive. Similarly, when you compute a neg score, you will multiply the

neg scores of the individual words, and the overall probability that a given comment might be negative. Compare the two scores to figure out which label, positive or negative, should be assigned to that comment.

Understanding When to use Naive Bayes

We've understood how we can use the Naive Bayes algorithm to solve a classification problem. It's actually useful to understand why the Naive Bayes is called naive. In the Naive Bayes algorithm, when we compute the pos score of a comment, we use the pos scores of each of the words independently. Each word contributes independently to the pos score of the overall comment. Here, we are not having any term to account how the pos score is affected due to a pair or more words appearing in the same context. For instance, if there were a phrase appearing inside a sentence, does that phrase add more towards the positivity or negativity of that sentence? This is not really considered inside this algorithm. The assumption that each word independently contributes to the pos or neg score of a comment, is the reason why the Naive Bayes algorithm is actually called naive. In general, the Naive Bayes algorithm assumes that each variable independently contributes to the final outcome, and interruptions between those variables are not considered. On the face of it, this seems like a pretty big assumption to make, and you might think that, because of this assumption, the Naive Bayes algorithm is not really that powerful. This is not true. The Naive Bayes algorithm is actually very robust, and ends up giving excellent results for many classification problems, especially in the cases where you use a small amount of training data, or if you don't have a lot of information or domain knowledge to create very relevant features. You can get good results with The Naive Bayes algorithm, even if the conditions of your machine learning solution are not perfect.

Implementing Naive Bayes

Let's take a real data set and apply the Naive Bayes algorithm on it. The data set that we'll be using, is called the sentiment labelled sentences data set. This is a public data set made available by researchers at the University of California. You can download this data set from the link that is shown right now. And when you download this data set, you will actually have three files in that data set, because this data set has sentences from three different sources. It has reviews from IMDB, Yelp, and Amazon, and all of these reviews have been typed as positive or negative, by the researchers. Each line in that data set, represents a review, and a label, zero or one. Zero is for negative sentences, and one is for positive sentences. The review and the label are separated by a

tab character. Now let's switch over to Python, and we'll implement the Naive Bayes algorithm on the sentiment labelled sentences data set. This implementation is done using the Scikit-Learn Module, so you will need to make sure that you have the Scikit-Learn package installed in your Python environment. Let's start by going to the University of California URL that I shared earlier, and downloading the data folders. From this, we will get a zip file that contains three text files, one each for Amazon, IMDB, and Yelp. It's useful to make a note of where the zip file has been downloaded, and its contents have been extracted. The Python environment that we're going to use, is a Jupyter or IPython notebook. You can download Anaconda, which is a Python distribution, and it will set up IPython notebook for you pretty simply. So just download and install Anaconda for your operating system, and open up an IPython notebook. Another advantage of using Anaconda is that a number of Python modules for data analysis, such as numpy, scipy, scikit-learn, and pandas, all come pre-installed for you. We need to start by reading the data from the text files which contain our label sentences. We'll do this by opening up the file, and then reading the lines in that file into a list. Textfile. read becomes a string with all the data in the text file. Then we split it by the new line category, to get the list which has each line as one element of that list. At the end of this, we have a list called lines, in which each element is a string in which we have a review, and a label for it, either one or zero. The review and the label are separated by tab. There are three files, imdb label text, yelp and amazon. We'll read the data from all these files, and open all the lines into the same list. We can take this list of lines and split up each element of that list, into another list, which has the sentence itself as the first element, and the label as the second element. At the same time, we are also making sure that any corrupted data, such as lines which don't have the label, or which are corrupted and don't have the tab separation, are not included in the data that we consider. We're doing this by checking if after the line has been split by the tab character, we have exactly two elements, and the second element is a label and not a blank. Now, if you print lines you'll see that each element is another list. One is a string, and another is an integer, zero or one as a string. Then we split our list into two separate lists. One is called training documents, which has only the sentences themselves. For this we just pick the first element out of each individual element in lines. Similarly, we create another list which has only the second element, from each individual list, the label as an integer, one or zero. We need to take the training documents list, and take each document and represent it, using numeric features. But we can _____ representation. Scikit-learn has a built in object called count vectorizer, which will do this for you. All you need to do is to instantiate a count vectorizer, and then use the fit transform metric, to convert the training documents into the list of numbers, which represent the frequencies of words in those documents. If you try and print train documents now, you will see that it is actually a matrix, a numpy matrix. The number of columns

is 5, 155. This is the distinct number of words that are present in our entire training data set. 3, 000 is the number of rules, which will be the number of documents or sentences, in the training data set. Each document in the training data set has now been converted into a tuple of 5, 155 numbers. Out of those 5, 155 numbers, many of them will be zero, because many words would not be present in the sentence. So if you try to print one training document, it will print the tuple by printing only the elements, which are one, leaving out all the other elements, which are zeros. Next, we move on to the training phase. Scikit-learn actually has built in objects called many different, classification algorithms, such as the Naive Bayes algorithm and support vector machines as well. Bernoulli NB is the object that will help us to perform Naive Bayes classification. We just need to instantiate Bernoulli NB classifier, and use the fit method, for that classifier. The fit method takes two arguments. The training data set represented using features, and the labels for that training data set. So we just fasten the training documents, which we have represented using down sequences, and training labels through the fit method. This results a classifier object which has a predict method that can take any new instance and classify it. So now we are in the test phase of our classification. We want to take a new sentence and classify it as positive or negative. All we need to do is to call the predict method, for the classifier object, and pass it a sentence. We have passed the sentence, this is the best movie. Do note that the sentence has to be passed as a tuple of numbers are in the term frequency representation. So once again we use the count vectorizer, to actually do this conversion for us. The predict method returns the label, whether it is a positive or negative sentence. Since this is a positive sentence, the predict method returns one as the result. You can pass it another sentence, like, this is the worst movie, which is a negative sentence, and you will see that the result becomes zero. So, we've built a Naive Bayes classifier that can take any sentence and classify it as positive or negative.

Detecting Ads Using Support Vector Machines

Let's go back to the problem of ad detection. You want to build a browser extension that can look at an image, and classify it as an ad or a non-ad. Let's go to and see how we can solve this problem using the support vector machines algorithm. We've already seen the problem statement. It's to take an image and classify it as ad or non-ad. The next step would be to take an image and represent it using numerical attributes called features. To represent an image, using a set of numbers, we can use attributes of the image, like the height or the width of the image. We can use attributes like the page URL, image URL, the page text and image text, which might indicate whether this is an ad or not, and convert these text attributes to numeric attributes. Let's move on to the training phase. Now we have every image in our training data, which is a large

data set of images, already labelled as ad or non-ad, represented using numerical features. Let's say each image is represented using N numbers. These images can be represented as points in an N-dimensional hypercube. An N-dimensional hypercube, what's that? This is pretty important to understand, and actually used in understanding a lot of different machine learning algorithms. To understand an N-dimensional hypercube, let's go back and understand basic shapes. A line is a one-dimensional shape. We know that every point on a line can be represented using one number, which is the distance of that point from the origin of that line. A square is a two-dimensional shape, and we know that every point inside a square can be represented using two numbers, the distance from the X axis, and the distance from the Y axis. Extending this further, a cube is a three-dimensional shape. And any point inside a cube, can be represented using three numbers, the distances from the X axis, Y axis, and Zed axis. Just extend this logic further. Even though it's difficult to imagine, an N-dimensional hypercube is some sort of physical representation, in an N-dimensional space. Each point inside that hypercube, is represented using a set of N numbers. Now since we've taken all the images in our training data, and represented them using tuples of N numbers, they can be represented as points in an N-dimensional hypercube. So here are all those images represented as points visually. All the points which are marked green, are images which are ads. All the points which are marked red, are non-ad images. In the training phase, the support vector machines algorithm will find the boundary that neatly separates these two sets of points. On one side of the boundary, you will only have ad images, and on the other side you will only have non-ad images. A hyperplane is just a surface that acts like a boundary in an N-dimensional space. In a three-dimensional space a plane is a boundary, or a hyperplane. With this, we are done with the training phase of the support vector machines algorithm. Then we move on to the test phase. In the test phase, given any new image, you should be able to tag it as ad, or non-ad. So, let's say we get a new image, which is represented by the orange point over there. We'll just check which side of the boundary this image falls on. If it falls on the non-ad side of the boundary, it is a non-ad. On the other hand, if you get an image which falls on the ad side of the boundary, you will tag it as an ad image. So this is how the support vector machines algorithm classifies any instances based on the boundary that it finds in the training phase. Support vector machines can only be used for binary classification. That is when you only have two labels to be classified into. Like in this case, we have ad and non-ad. This is not the case with the Naive Bayes algorithm, where you can classify into any number of categories.

Implementing Support Vector Machines

Let's take a public data set and implement the support vector machines algorithm. The data set that we're going to use, is the Internet Advertisements Data Set, from the University of California Machine Learning Repository. The link to download this data set is being shown on screen right now. This data set represents images. Each image in the data set has already been represented using features or numeric attributes. In fact, each image is represented using nearly 1500 numeric attributes, the descriptions of which are being shown on screen right now. Along with this, each image is also labelled as ad or non-ad. This is a snapshot of what one row in this data set looks like. We've got those 1500 plus numeric attributes, and at the end, we have a label saying either an ad image or a non-ad image. Let's now move this data set into Python, and use it to implement the support vector machines algorithm, for image classification. Let's start off by downloading the internet advertisements data set from the URL that I shared with you earlier. You can download the data set by clicking on the link for the data folder, and then downloading a file called ad dot data. You could also choose to download the zip file, which contains the documentation for the data as well. But the file that we will actually be reading, is the file called ad dot data. We'll be reading this file using a library called pandas, and then manipulating it using data frames. Data frames are table-like data structures, which make data analysis pretty simple. You can do a lot with one line of code. We'll also use the NumPy library which helps us do some numerical computations. If you are using a Python distribution like Anaconda, both of these libraries will be pre-installed for you, so all you need to do is import them into your IPython notebook, then to read a file, pandas has a function called read dot CSV. So, you can just read the file by giving the part of the file, the separator of the columns in the file, and specifying whether there is a header or not. The data from the file has been read into a data frame called data. We can use the head record to get a glimpse of the data within this file. As you can see, there are 1559 columns for this data. Each row in the data represents one image, which is tagged as ad or non-ad in the last column. Column zero to 1557 represents the actual numerical attributes of the images. Some of these attributes might be missing, as you can see, in row 10, there are three values missing in the first three columns. We need to do a bit of data processing, to take care of these missing values. First we'll write a function that will check whether a given value is a missing value, and if yes, it will change it to a NumPy value called not a number. The reason we're doing this, is because pandas data frames have a metric called dropna, which will drop any rows which have missing values, and the way pandas recognizes missing values, is if the value is not a number. So this is the function that we've written, that will take any value, it will try to convert it to a float. If it cannot convert it to a float, that means it is some sort of corrupted or missing value. In such a case, it will return not a number. In the other cases, it will return the float itself. This function can only be applied on a cell of a data frame. But there is an easy way to take a function,

and apply it to every member of a pandas data structure, and that is using the apply method. We'll use the apply method in two steps. First we will use it to apply the toNum function to every cell in a column, then we'll take that function that can apply toNum to every cell in a one column, and apply that to every column in the data frame. So the seriestoNum function will take the toNum function, and apply it to every cell within that column, then we take that seriestoNum function, and apply it on every column, except for the last column in the data frame. The last column in the data frame is the labels column, ad or non-ad, and we'll need a separate function to deal with that column. We just take the first 1558 columns, and apply the seriestoNum function on those columns, and once that is done, you will be able to see that wherever we saw missing values earlier, for instance, in row 10, now you will actually see not a number in place of the missing values. We're almost done. Now if we use the dropna method on this data frame, it will subset the data frame to only those rows which don't have any missing values. So now this data frame does not have a row 10 anymore, because that row had missing values. We now have our training data ready. We also need to get the training labels ready. So for that, let's go back and write a function that will take the last column in our data frame, and convert it to an integer. If the column says ad, it will return one, otherwise it will return zero. We'll use the apply method to apply this function onto the last column of the data frame, and we get a series called, training labels. In order to make sure that the training labels line up, to the training data, which means that each row in training labels should line up with its corresponding row in training data. We use the training data's index attribute, which will contain only the row numbers that are actually present in the training data. After all that data set up we're actually ready to perform support vector machines ad detection. For that, we'll import linear SVC, which is a classifier object, that can perform support vector machines, from the scikit-learn module. Like other classifiers in the scikit-learn module, linear SVC has a fit method, which can be used to perform the training phase. For this fit method, we need to pass in a training data set, and a training label series. We're only using row 100 to 2300 of the training data set, for this training exercise, because we want to leave out some of the training data, for the test phase. In general, when you do a test phase for a classification algorithm, you would take a new image, which has not been seen in the training phase, and pass that through the classifier that was built in the training phase. However, in our case, we only have the images from this complete data set we downloaded from UCI, so we need to keep some of those images aside, to act as new images, that the classifier has not seen before. After the training phase, we get the classifier object, CLF. For this classifier object, we can use the predict method, to pass in a new image, and check out whether that image has been classified correctly as ad or non-ad. Here, we are passing in row 12 from the training data, which if you go back and check, will be actually an ad, and the classifier correctly predicts that it is an ad by

returning one. Similarly, we can check out another image. This image, in the last row of the training data set, is actually a non-ad, and the classifier is able to correctly predict that as well, by returning zero. So that brings us to the end of this module, of solving classification problems. Let's summarize what we've learned. We've understood the Naive Bayes algorithm, and how to use it to solve sentiment analysis. We understood why the Naive Bayes algorithm is called naive. We understood what the support vector machines algorithm is, and we saw how it can be used to solve ad detection, or image classification.

Predicting Relationships between Variables with Regression

Understanding the Regression Setup

Welcome to this module on regression problems. Regression is really helpful when you want to understand relationships between different variables. In this module, we'll learn how to recognize regression problems in different fields, whether it is quant trading or demand forecasting. We'll also understand how to set up a regression problem and the dependent and independent variables, which comprise the regression problem. You'll also be able to understand the difference between classification and regression and when each of these techniques should be used. To give you an idea of where regression can be used, here are a few different problem statements of machine learning regression problems. What will be the stock returns on a given date? If the waiting time increases, how does this affect customer satisfaction? And what will be the sales on a future date? So there are two themes in these problems. One type of regression problem statement deals with computing some kind of continuous value. For instance, if you wanted to compute the value of stock returns on a given date or the value of sales on a given date. The other type of regression problem statement deals with quantifying the relationship between two or more variables. For instance, we have wait time as a variable, and we want to understand how customer satisfaction will be affected if the wait time changes. Let's take the example, where we want to compute stock returns on a given day and try to understand how regression works. Stock returns on a given day might depend on a number of different variables. Here are a few examples of the different variables that might influence stock returns. The day of week, the day of the

month on which we're trading, and the daily stock returns of that stock in the last one week. These are all variables that might affect stock returns on any given day. Let's say we knew the values of these variables. Could we compute the stock returns by using some sort of mathematical function that combines these variables? That's exactly what regression does. Regression will find a function that can peak the input variables and compute the stock returns. That function quantifies the relationship between the input and the output. The output is called a dependent variable, and the input variables are called independent variables. Any regression problem can usually be expressed as finding some function that relates two sets of variables. Regression usually starts by assuming a particular form for that function that it's trying to find. If the mathematical form is a linear equation, then it is called linear regression. If the mathematical form is a polynomial, then the algorithm is polynomial regression. Otherwise, if the mathematical form is assumed to be non-linear, an algorithm called non-linear regression is used. So all of these are different regression techniques based on the form of the function that the regression needs to find. Of these, linear regression is by far the most well-known and commonly used regression technique.

Forecasting Demand

Let's look at the problem of demand forecasting. Many businesses need to estimate what their sales might be at a future point in time. This is actually a crucial exercise and has a lot of impact on the efficiency and the costs of the business itself. A retail business, for example, will use the results of demand forecasting to decide whether it should scale down its inventory or buy more. If it thinks the sales are going to be very high in the near future, it will replenish its inventory. Otherwise, it will scale down its inventory. A manufacturing business, for example, will find demand forecasting very important. How much it should produce in each cycle will help determine things like how much it should order from its suppliers and when, what kind of resources it needs to align them, in terms of labor and transport, and how to plan all of this out. So as you can see, demand forecasting is a crucial question for many businesses. The problem statement here is to predict what the sales might be at some point in the future. The sales at that future time might depend on the sales in the previous cycle, whether it's a week, month, or quarter. This captures a base level of sales, as well as any upward or downward trend in the sales that is occurring at that moment in time. Another important variable might be the expected marketing spend. Depending on any promotions or ads that are going to run in that period, the sales might go up or down. Holiday seasons, such as Christmas or Black Friday, have a big impact on what the sales might be during that period. So on one side, you have variables like the sales in

the previous week, marketing spend, and holidays. And on the other side, you have the sales in the future week. You have a dependent variable. You want to predict using some independent variables. You want to find a function that relates these two sets of variables. And that is exactly what regression will do for us. Demand forecasting is a perfect example of a problem where you would use machine learning regression.

Predicting Stock Returns

Let's go to the world of quant trading. High risk, high reward, this is a maxim that is heard quite often in the business world. And it's actually quantified in a quant trading model called the capital asset pricing model. This model is an equation that relates the returns of a security with the returns of the overall market. And it's generally used for pricing risky securities. Let's parse this equation. On the left-hand side, you have the returns if you invest in a particular security. This equation says that this returns of a particular security depends upon the risk-free rate of return. This is the base rate of return from a risk-free security. Normally, treasury bonds and T-bills are considered risk-free securities from which you can expect a base level of return with very low risk. However, since the security that we're pricing is actually a risky security, the returns from that security will have a premium over the base risk-free rate of return. That premium is computed using that term on the right. This term comprises of a value called the volatility of the security, which is a measure of how risky that security is. To find the premium, multiply this volatility by the expected returns of the overall market over and above the base risk-free rate of return. This particular equation has this value called beta, which is an important measure that helps us understand how risky a given security is. And it can tell you how much risk a security adds to a portfolio of securities. If you're not interested in quant trading, and all of that was a little too much for you, all you need to remember is that beta is an important measure that a lot of folks are interested in, and this particular equation quantifies the relationship between stock returns and the overall market returns using beta. So if you used regression to quantify the relationship between stock returns and market returns, you could find the value of beta. So this is another example of where regression could be useful. In general, whenever you want to find the measure that represents a relationship between two variables, that is when regression comes into play.

Detecting Facial Features

Let's consider the problem of detecting facial features. Here is a picture of a face. As you can see, that is just my face. And as a human, you can look at this picture and identify where the eyes,

nose, and mouth on this face are. How would you teach a computer how to identify where the eyes, nose, and mouth are present in this particular image? This is a very interesting problem to solve, and it's very useful as well. Because if a computer can easily identify these different facial features in an image, you can use that to build systems such as facial recognition systems or virtual dressing rooms where you can try on a pair of sunglasses from an online store without ever stepping out of your house, or a camera being able to auto-focus and auto-capture a photo whenever it detects a face. The problem statement here, is to find the coordinates within this image of the important facial features. For example, what are the x- and y-coordinates on this image of the left eye center, or the right eye center, or of the nose, and so on? The coordinates of these features actually depend upon the relative position within the picture. So the overall size of the picture matters. And also, what are the surrounding pixels like for that particular pixel? We can actually compute the position of each individual facial feature using one separate regression problem. So you can set up a bunch of regression problems to find out each of those individual features. Here is one such regression problem. Find the coordinates of the left eye center given the size of the picture and the greyscale value of each pixel. The left eye center coordinates are the dependent variables, and the size of the picture and greyscale value are independent variables. Using regression, you can find a function that relates these two sets of variables. You would set up such a regression problem for every individual facial feature that you want to find, and that's how you can solve facial feature detection using regression.

Contrasting Classification and Regression

Classification and regression are actually similar in many ways. Let's go through what are the different ways in which classification and regression are similar. If you'll recall the typical classification setup, it involves defining a problem statement, classifying some object into a set of categories, representing the data points that we want to be classified as features or numerical attributes, feeding a set of training data for which we already know the output to an algorithm so that a model can be trained, and then finally, testing that model by giving it new instances to classify. It so happens that regression actually also has the same four steps involved in its setup. Although we haven't defined the regression flow in this particular way yet, we can see the similarities if we just take a closer look. The problem statement in classification was to assign subcategory or label. Regression also has a standard problem statement, which is to compute some continuous value from other variables. So just like classification, regression, too, has a standard form for the problem statement. And usually, when you start with regression, you would have to define this problem statement for your particular scenario. The next step in classification

and in regression is to identify features or numerical attributes to represent your input. In classification, the input object which has to be classified is represented using numeric attributes, and those become the features. In regression, the independent variables are the inputs for your regressors, and those are the features in your problem. The only thing here, is that in classification, all the attributes or features pertain to one particular instance or object that is being classified. But in regression, the features need not be related to one particular object of instance. Just like classification, regression also has a training phase. In classification, you take objects that are already classified and feed that as training data to build a classifier. In regression, you take a data set for which the input and output is already known, this is the training data, and you use it to quantify the relationship between the independent and dependent variables. So you cannot perform regression if you do not have historical data or training data. The regression algorithm will find the relationship between the input and output variables by learning from the training data. Therefore, regression is also a form of supervised learning, just like classification is. To recap, whenever you have a machine learning technique that involves an explicit training phase, then you are said to be performing supervised learning. And lastly, both classification and regression have a test phase as well. The only thing is that, in classification, a label is assigned out of a predefined set of categories. In regression, a continuous output value is determined given the independent variables. So the overall setup of a regression problem is very familiar to a classification problem. So what are the main differences? How do you differentiate really, between a classification and a regression problem? The main difference is that, in classification, you are looking for a categorical output. The output of classification is a choice out of a set of categories. In regression, however, you get a continuous output. The value that is computed at the end of regression, the dependent variable, so to speak, is a continuous value. When you're not exactly sure which technique it is that you want to use between these two, then look at the output that you are looking for. If the form of the output is categorical, you need to perform classification. If the form of the output is continuous, then go for regression. Another important difference between classification and regression is how the relationship between the input and output is expressed. In general, when you build a classifier, it is a black box. You don't really care what happens inside the black box. The classifier will just take some input and give you the right category it should be assigned to. Regression, however, starts with an assumption about what the form of the relationship between the input and output will be. And usually, this is a mathematical function. So if you are looking for a mathematical function that actually quantifies the relationship between your input and output, then choose regression as the technique that you need to use. We've learned how to recognize regression problems in different fields, whether it's quant trading, demand forecasting, or image processing. And we've also learned how to set up a

regression problem using the dependent and independent variables for these different examples. We know how to differentiate between classification and regression and when it is appropriate to use each of these techniques. In the upcoming module, we'll be looking at how to solve regression problems, specifically, using a technique called linear regression.

Solving Regression Problems

Introducing Linear Regression

Welcome to this module on solving regression problems. We have learned how to recognize regression problems in the world around us. In this module, what we'll do is take one specific regression algorithm, and apply it to a specific regression problem. We'll understand how linear regression, which a regression technique, can be applied to find the beta of a stock. We'll understand how to use the Stochastic Gradient Method for linear regression. This is one of the sub techniques in linear regression. Then we'll understand how to tweak some of the parameters of the Stochastic Gradient Descent Method so that we can get better performance. Then we'll implement linear regression in Python. This is the form of regression, this is something that we have seen before. You have a dependent variable and independent variables, and you would like to find the function that relates these two sets of variables. The objective in regression is to find this function that can quantify the relationship between the dependent variable and the independent variables. Regression does this by assuming a specific form for this function that you want to find. The regression technique used depends upon the form that you assume the function to take. If you assume that the function is linear, then we use the technique called linear regression. As the name suggests, linear regression means that the function is assumed to have a linear form. That form would look something like what you see on screen here. The function is a linear combination of the independent variables. The b's here are actually constants, and the whole objective in linear regression is to find the values of these constants. The way linear regression does this is, it solves for the values of those constants by using past data. So you would take past data where you have the values of both the dependent and independent variables, and then solve for the values of the constants. Then in the future, you can just find out the value of the output, using these constants and the input variables. These constants that you find are actually called coefficients, and they quantify how much each independent variable has

an impact on the output variable. Let's consider the example of demand forecasting. The dependent variable is sales, that's what you want to predict, using values like the marketing spend, and the sales of last week. Let's say you performed linear regression, then you would end up with an equation that looks like this. On the left-hand side you have the dependent variable of the value that you actually want to compute after regression. On the right-hand side, you have the independent variables. The dependent variable is a linear combination of the independent variables. These are the coefficients that are found using linear regression. The values of this coefficients tell us how much each of these independent variables contribute to the final value of the dependent variable.

Applying Linear Regression to Quant Trading

Here is an equation representing a well-known financial model called the capital asset pricing model. If we rearrange this equation, just taking the risk-free data for r on the right-hand side to the left-hand side, we have another form of the capital asset pricing model, but this is a little interesting, because it looks very much like a linear regression equation. On the left-hand side you have the dependent variable, which is expressed as a linear function of one independent variable on the right. When you just have one independent variable, this particular equation is very much like the equation of a line passing through the origin. So if you had to see this visually, then you would have a line passing through the origin. The y-axis is represented by a dependent variable, which is the returns of a given stock, and the x-axis is represented by an independent variable, the overall returns from the market. We take both of these returns over and above the base risk-free date of return. Now take this line and compute the slope, the slope of that line will be a measure called beta, which represents the risk of that stock that we're looking at. If you actually plotted past historical financial data for a stock and for an overall market, then you would have data that looks like this. The line that we saw before is actually the best fit line for this set of data points, and linear regression is the technique that will help us find that line that is the best fit. If you are looking to find the beta of any stock, then all you need to do is perform a linear regression between that stock's returns and the overall market's returns, then the slope of the line that the linear regression will find will be the beta that you're looking for. This particular form in which we have only one independent variable, is called a simple linear regression with one variable. You can perform linear regression with multiple independent variables as well, and that will be called multiple linear regression. Now let's say you find the beta. Then, you can actually take the market returns on any given day, and multiply by beta to find the predicted value of the stock's returns. That predicted value will be some point on this line that we have found. There will

be a problem though. The actual returns for that stock on that day would not exactly lie on that line, but slightly away from the line. So there would be an error between the actual returns, and the predicted returns using this equation. So the error, visually, is the distance between the actual point representing the returns, and the line itself. The error represents the leftover variations of the parts of the dependent variable, but are not explained by the variations in the independent variable. This error is also called a residual. Whatever the regression technique you use, or whatever the form of the mathematical function relating the independent and dependent variables, there would be an error between the actual value of the dependent variable, and the value computed using your function. The goal of any regression algorithm, and the goal of linear regression as well, is to minimize this error for a set of data points that we have, called training data. Simply put, the objective of linear regression, if to find the line that will minimize the error between the actual value and the predicted value for the training data set.

Minimizing Error Using Stochastic Gradient Descent

We've mentioned that linear regression will find a linear equation to relate the input and output variables. Linear regression does this by trying to minimize the errors between the actual value of the output, and the predicted value of the output, by using the linear equation for a training data set. So this is set up as a standard optimization problem. There are several techniques in optimization for minimizing a value given certain constraints. One such technique is the Stochastic Gradient Descent technique. Gradient descent, as the name suggests, has to do something with decreasing slope. The goal of gradient descent is to minimize a function for the error value. One of the common functions used to measure error is the squared value of the error. Let's say your training data set has n data points, and you computed the error for each of those data points. Then the mean squared error of all those data points is the total error function that we want to minimize. Mean squared error is just one way of measuring the overall error. There are several other functions which might be used to measure error. In this case, let's talk about the mean squared error. This mean squared error is actually a function of the slope and intercept of the line that we are going to find. So for different values of the slope and intercept you would get different values of the total mean squared error. Let's say the graph on the right represents the idle value for different values of slope and intercept. Each point on that graph on the right represents the value of the error for one particular value of slope and intercept. So here's how the Stochastic Gradient Descent method works. It will take a random value for slope and intercept and choose that as the initial value. Let's say this point represents the initial position of the slope and intercept, and the current value of the error function. Since our goal is to minimize error, we

actually want to move from this point towards a point which is lower on the graph as we see it. The further downwards you move from this point towards a point which is very low in the graph, the more your error is minimized. Now at this point, you can take a partial derivative, with respect to the slope and intercept, and find out what the slope of this graph itself is. You would find the slope at that initial position, and move slightly downwards in the direction of the slope. In the direction of the gradient, that is why we are saying gradient descent. Once you have move downwards, you have got a new point. You have a new value of the error, and you can compute new partial derivatives for the slope and intercept. Again, you can move slightly downwards in the direction of the gradient. You would repeat this process until you reach a minimum, a minimum for the error itself. Practically, it may not always be possible to go on until you reach a minimum. You would stop either after a certain number of iterations, or when the error doesn't change much any more in each situation. At that point, the slope and intercept that you have define a line, and that line will be the best fit line that relates your input and output variables. This is how you can use the Stochastic Gradient Descent technique to find the best fit line. Usually data science friendly programming languages, like Python, R and Spark have built-in libraries, which implement the Stochastic Gradient Descent method for you. However, they expect you to be able to play around and tweak with either the number of iterations, or the step size that you would take in each iteration. How much would you move down the slope in each iteration. Both of these parameters are things that you can tweak when you set up a Stochastic Gradient Descent regressor in any language, like Python, R, or Spark.

Finding the Beta for Google

Here's the equation that we've been talking about so far in a linear equation example. Let's say we wanted to use this equation to find the beta for Google. So let's basically use the CAPM model to find the beta for the stock of Google. Yahoo! Finance actually has a value for the beta of Google, and if you look up Yahoo! Finance it will say that the value is 1. 03. So let's see how close we can get to that value using our linear regression model. To apply linear regression we will need to get historical data for the returns of Google, and the returns of an index or an overall market. To represent the returns of the overall market, which is the independent variable on the right-hand side, we'll use the NASDAQ index, which is a composite of most of the well known stocks that trade on the NYSE. So let's see how to find beta using historical data, and what are the steps involved in implementing this in Python. First we would download historical prices for Google and NASDAQ from a financial site, such as Yahoo! Finance. You can just search for GOOG on Yahoo! Finance, that is the symbol for the company Google. You can also search for carat IXIC, which is

the symbol for the NASDAQ compositing mix. Here we're download about six years of data from 2010 to 2016 at a monthly frequency. We'll get two files, one for Google and one for NASDAQ. Each will contain the stock prices at the end of each month for Google and NASDAQ. The next stop is to take these prices and convert the prices to returns. The returns would be what you would get if you invested in a stock at the beginning of the month and then sold it at the end of the month. We can use the adjusted closing price, which is the last column in the CSV file to find the returns. The monthly return percentage would be the difference in price from the beginning of the month to the end of the month, divided by the price at the beginning of the month. So this would give us a monthly return for Google and NASDAQ. Both of these would be two series of numbers. As we mentioned at the beginning of this video, the Google returns is our dependent variable, and NASDAQ returns is our independent variable for our regression. But before we perform the regression, we need to remove the base risk-free rate of return from each of these variables, and then we can perform our regression. The risk-free rate of return is computed using the yields of five year Treasury bonds. You can search for the symbol caret FVX on Yahoo! Finance, and then download the data for the same period, 2010 to 2016, For the same frequency, monthly. This would give us another CSV file. That CSV file would have a column called adjusted close, which represents the yield percentage. We need to divide the yield percentage by 100 to actually compute the yield. Now take the Google and NASDAQ returns that you already computed and subtract the yields of five year Treasury bonds from both of those series, then you will finally get the Google returns minus the risk-free rate, and the NASDAQ returns over and above the risk-free rate. These are the two series that we will use in the linear regression algorithm. Finally, we'll use the Scikit-Learn module in Python, and we'll use a regressor called the SGDRegressor. This is the linear regression with Stochastic Gradient Descent. So this SGDRegressor will just provide the Google returns and NASDAQ returns as the dependent and independent variables.

Implementing Linear Regression in Python

Let's look at a demo implementing linear regression in Python. We'll use the same example that we discussed just now, where we will compute the returns of Google and NASDAQ and five year Treasury bonds, and use them to compute the beta of Google. We'll do this by regressing the returns of Google against the returns of NASDAQ using linear regression with Stochastic Gradient Descent. We'll be using the Pandas and NumPy modules in Python to pre-process our data. We'll perform all the steps that we explained already. We put all our pre-processing logic into a function called read file. This does the job of reading a file from Yahoo! Finance, downloading the

prices to returns, and returning a data frame which only has the monthly returns for that stock. In case it's a Treasury bond file that's being read instead of a regular stock or index, then this particular function will return yields divided by hundred, which is exactly what we are looking for. Here we are using this function to read the financial data for Google, NASDAQ, and the five year Treasury bonds. We're now ready to perform our regression. We'll import a regressor object from Scikit-Learn. Here are two examples of regressor objects that Scikit-Learn provides, SGDRegressor, which is what we're going to use, and linear regression, which is a straight linear regression using another method called Ordinary-Least Squares. We want to use the Stochastic Gradient Descent method, so we'll use SGDRegressor. We'll instantiate in SGDRegressor. SGD requires two parameters: the step size that it should move in each iteration, and a maximum number of iterations, in case it does not reach a global minimum before this number of iterations. In our Kappa model there is no intercept for the line, the line just passes through the origin, so we can say that the intercept is actually not required in this linear regression. So we set the fit intercept to false. Now all we need to do is to call the fit method, this is the training step, it will perform the regression and find the linear function that relates Google returns and NASDAQ returns, and after the regression is done, we can ask the regressor to share the coefficient, which will actually be the value of beta that we are looking for. What this means is that if you wanted to compute Google returns over and above the risk-free rate at any point in time, then you could take NASDAQ returns at that point in time and multiply it by. 9 and you would get the Google returns. That's the meaning of this linear regression and this coefficient. That brings us to the end of this module on solving regression problems. We've understood the Stochastic Gradient Method for linear regression, and we've seen how to set these parameters in Python and implement linear regression to find the beta of a stock.

Recommending Relevant Products to a User

Appreciating the Role of Recommendations

Welcome to this module on recommendation problems. In this module, we'll be talking about the category of machine-learning problems called recommendations. We'll understand why personalized recommendations have become so important in today's business world. We'll be able to predict user-product ratings using collaborative filtering and mix. We'll understand how to

find hidden factors that influence user-product ratings. And finally, we'll implement the alternating least squares algorithm in Python to find movie recommendations. Recommendations are actually ubiquitous these days. If you visit any e-commerce website like Amazon, you would see recommendations for products that you might like and want to purchase. If you visit any service like Netflix or a music-streaming service, you would see recommendations for movies or artists that you might like based on what you seem to have liked before. Services like Gmail can prioritize your inbox and automatically tag emails that you might find important. All of these are examples of personalized recommendations. All of these services are trying to predict what you might need or like, even before you know it, yourself. Whether it's the homepage of the newspaper that you like, whether it's your Facebook news feed, or whether it's your inbox organization, all of this is personalized to you based on needs. Personalization has become a huge trend across all these online services, because the big advantage of something only online stores can do. Webpages and online services used to be static, like offline stores. If you walk into an offline store, everybody who walks in sees the same thing. Layouts and product placements in offline stores are usually designed to appeal to a majority of users, and not for specific users. Now, imagine this magical world where, if you walk into a store, the store automatically rearranges itself so that you would see the design that appeals to you and only the things that you require show up right in front of you. Clearly, we don't live at Hogwarts, so this is not really possible in real life. But it is possible for an online service to do. An online store can personalize its look and feel so that it appeals to a specific user. The store can be personalized based on the user's preferences, needs, and what they are currently searching or looking for. In fact, personalization is not just something cool that online stores can do. It's actually helping these stores and services solve two really important problems. The problem of discovery and engagement. Unlike offline stores, online stores normally have really huge catalogs. If you look at Amazon, it has millions of books. iTunes might have thousands of songs, or Netflix might have thousands of videos. And users are generally only going to look at the homepage or a few other pages, which are easy to find. They need help finding what they're looking for, and you need to make sure what they're looking for is easily available to them on the homepage or on pages that they might encounter without too much trouble. Sometimes, these recommendations might help users find things that they didn't know that they were looking for. The other problem personalization helps to solve is engagement. The more time users spend on a website, that's engagement, the more likely they are to open their wallets and buy something, or do something on your website. So, if you personalized your website to meet the needs of the user, then that makes it more likely that they will be engaged, and spend more time on your website. This is the problem that machine learning algorithms are

trying to solve. They're trying to increase the discovery and engagement by providing personalized recommendations to users based on their particular tastes and needs.

Predicting Ratings Using Collaborative Filtering

We've seen how important personalized recommendations can be for any online service. Now, let's look at a few different examples of the kind of recommendations that you might find on an online service, and we'll see that there is a common problem to be solved across all of those different types of recommendations. Here are the top 10 movie picks for you. This is a type of recommendation that you might come across when you use any video service like Netflix or YouTube. Here is a quick way that you could actually solve this problem. Let's say there was a way for you to find a preference score or a rating for every product by every user. Then, all you need to do to find the top 10 movie picks for a user is to sort the movies in descending order based on the ratings, and pick the top 10 movies that these user hasn't yet watched. So if you knew what would be the rating for every movie, by every user, then you would be able to find the top 10 picks for any particular user. You might have come across recommendations sometimes which are based on browsing history, which means that the recommendations are shown to you because you have seen some other products. Most often, these might be shown under the name: because you shopped for, or: because you watched this, and so on. Here, we can look at the number of views of a product by a particular user as an implicit rating, or a preference score. Similar to what we were talking about in the previous example. Now, if we were able to predict the number of views that a user might give for a product in the future, based on what they've already viewed, we could basically compute the rating, or the number of views, for all products, for all users. Then all we need to do is to pick the top 10 products for this particular user, and show them as recommendations based on browsing history because here, the rating is derived from the number of views of a product. Oftentimes, when you're looking at an item on an online service, you might see a recommendation that says that other users who looked at that item also liked some other items. Once again, to solve this problem, you would compute the ratings for all products, for all users, then all we need to do is to find products which were highly-rated by users who had bought the product which is under consideration. So we would subset all the computed ratings to users who have bought this particular product, then pick the 10 products with the highest ratings, within this subset. So, the common theme across all of these three examples is that if we are able to compute the ratings for all products, by all users, then we can find many different types of personalized recommendations based on that information. Clearly, the problem here is to be able to predict the rating for a product by a user, even if that user has never

indicated any previous preference for that specific product. This is exactly where collaborative filtering comes in. Collaborative filtering algorithms are basically algorithms that look at users' ratings for products in the past and use that to predict a user's rating for some other products which they have not yet bought or seen. So collaborative filtering algorithms can be fed data like: user purchases, user browsing history, the clicks that a user has made, and ratings and reviews by the user in the past. And the algorithm will provide the ratings for every product by every user, which can be used to create recommendations like: top picks for you. If you like this product, you'll love that as well. If you bought this product, then you'll need this other product. There are other techniques which help us solve this problem of finding ratings for all products by all users. Specifically, content-based filtering is one example. But collaborative filtering is a general term for any algorithm that uses only past user behavior for solving this problem. In contrast, the content-based filtering technique, for example, uses product attributes as one of the inputs of the algorithm. Whereas collaborative filtering doesn't do any of that. It only looks at users' past behavior. The key premise in collaborative filtering is very simple. When you want to recommend products to somebody, then all you need to do is to find someone who is similar to that particular user and see what other things they have liked. Similarity, here, depends upon the opinion those users have about products. So if two users have the same opinion about some products, then they're likely to have the same opinion about other products, as well. And this is exactly the premise behind collaborative filtering. Collaborative filtering algorithms require training data. The training data is usually in the form of ratings or preference scores by some users for some products. So the columns in the data would likely be: User ID, product ID, and the rating that this user would have given this product. We're using product as a general term, here Product could actually be a book, a video, a movie, something that you're actually buying on Amazon, or even a news article or an email, as well. Collaborative filtering inside an inbox can help you prioritize your inbox. The rating used can be any measure which shows a user's preference for a particular product. This could be an explicit rating, which are ratings that the user has specifically filled, say on a one to five scale or a one to 10 scale. Either when they were buying or viewing some product on the store, or through some email survey. On the other hand, you might have implicit ratings, which are measures like the number of clicks, number of purchases, number of shares or likes, or the number of times a user has viewed a video or listened to a particular artist. Once you have such a data set which has users' ratings for a certain set of products, then you can feed this to a collaborative filtering algorithm and the algorithm will compute predicted ratings for all the products by all the users, and that's what will help us provide any different type of recommendation.

Finding Hidden Factors that Influence Ratings

We've been talking about collaborative filtering algorithms. Now, there is a subset of collaborative filtering algorithms which basically work by identifying hidden factors that influence user behavior. As you know, the problem that collaborative filtering is trying to solve is to find ratings for every product by every user. So let's try and understand how latent factor analysis and finding hidden factors can help us compute user ratings. Here are some factors that might actual influence a user's preference for a particular movie. The genre of the movie. The popularity of the cast. The commercial appeal. Or the recency of release. All of these might be guiding a user to say that they like certain movies more than others. Now, to represent this in some quantitative way and then use it in an algorithm, what we might do is to take every user and rate them, based on the importance they give to these factors on a scale of one to five. Similarly, we would have to take every movie, also, and rate it on a scale of one to five. So, every user and every movie has been represented using four numbers: one score for each of these different factors. The use of four numbers, each, might be thought of as points in a four-dimensional space. Now, we've represented all the users and all the movies as points in some four-dimensional space. And here we have all the users represented in green, and all the movies represented in red. Now, if we take one user and look at the movies which are nearest to that user, those might be the movies that this user will have a high preference for. So if you want to find recommendations for this user, you would find the nearest neighbors to this user. Once you have represented all users and movies, in terms of influencing factors, this method of using hidden factors is a logical and intuitively understandable method for finding recommendations. But there happen to be quite a few assumptions that have gone into this method. First, that we know which factors actually influence users' preferences. Second, that we can quantify those factors for each user. And third, that we can quantify those factors for each movie. These are actually pretty big assumptions to have, and it's very rare that you would find a data set which actually has hidden factors and users and movies rated on all those factors. So while the idea of having all users and products or movies rated against different factors is pretty cool, it's quite difficult to actually have this data in real life. This is where latent factor analysis techniques come in. If you have a data set that has the user ID, product ID, and rating for a certain set of products, then you could take that, feed it to a latent factor analysis algorithm, and it will break it up into users, quantified in terms of some hidden factors and products quantified in terms of some hidden factors. Once you have this, you can find the nearest products to a particular user, and those would be the products that you would recommend to that user. So the reason why this is called latent factors or hidden factor analysis is because the factors are not known beforehand. We're not hypothesizing that there is a genre or

past popularity factor that actually influences users' ratings, but we are letting the user rating data tell you what the factors actually are. Once the factors are actually identified, they might turn out not to be something meaningful. They might turn out to be. There's no guarantee for that, though. The factors might have some meaning, like genre or cast popularity, or they might just turn out to be abstract factors with no real-life meaning. But just, somehow, users and products can be categorized based on those factors. So how do latent factor analysis techniques work? You would take this user ID, product ID, rating data, and represent it as a matrix. Each row in the matrix represents a user, and each cell represents the rating for some product by that user. Now, you can take this matrix and break it down into two separate matrices. A user-factor matrix, where each row represents a user, and a product-factor matrix, where each column represents one particular product. In each row of the user factor matrix, we have a user represented in terms of their hidden factors. In each column of the product factor matrix, we have a product represented in terms of the hidden factors. Let's say we call the user factor matrix P and the product factor matrix Q. And the original user/item rating matrix R. Each cell in R is actually the dot product of a row in P and a column in Q. For instance, here we have R-43, which is a rating by user four for product three. This rating is the dot product of the row P-4 and the column Q-3. So to find a rating for any product by any user, all we need to do is to take a dot product of the corresponding row in the user matrix, and the corresponding column in the product matrix. Notice the cool thing that's happening here. If you look at the user/item rating matrix, R, there are quite a few empty cells in there. This is because all users will not have actually rated all products, whether explicitly or implicitly. So we take only the ratings which we actually have, we compose that matrix, and from the decomposed matrices, we can compute what the ratings would be for any product. The same matrix, here, can actually be represented as a set of equations of this form. Where each rating is the dot product of a user vector and a product vector. Vectors are just rows or columns of matrices. Once you solve this set of equations, you have all the users and products quantified in terms of factors, which is exactly the problem that we set out to solve. So the problem that we want to solve is expressed as: find that set of factor vectors, PU, which represents a user factor vector, and QI, which represents an item factor vector, or a product factor vector, such that when you take the product of these vectors, the error between the product and the actual ratings that we have in the training data is minimized. So this is clearly set up as an optimization problem. We have a value or an objective function that we want to minimize, given certain constraints. Alternating least squares is one of the popular optimization techniques that is used to solve this problem.

Understanding the Alternative Least Squares Algorithm

We've mentioned how latent factor analysis can be used to solve the recommendations problem, and alternating least squares is an optimization technique that can minimize this error function between the actual ratings present in our training data and the user factor vectors and product factor vectors that we might find. RUI represents an actual rating for a user, for a particular item, and QI represents the item factor vector. PU represents the product factor vector. PU and QI, for all the different products and all the different users is what we're trying to find. The error function that we have is actually a sum of errors for all ratings, but if we concentrate on one rating, then the equation to solve is minimizing the error for one particular rating. This is actually a quadratic equation, which has two variables that we want to solve for. PU and QI. Obviously, the problem here is that we have one equation and two variables. So how do we solve this particular problem? What if we fixed the value of PU to some random value? Then we're actually left with a quadratic equation for the value of QI; we have one equation and one variable. And we can solve that pretty easily. On the other hand, if we fix the value of QI, then again we are left with a quadratic equation for the value of PU, and we can solve that equation easily. As the name suggests, alternating least squares basically does exactly this. Alternating between fixing the value of PU and QI. So first, it would fix the value of PU, and solve for the value of QI. Then, it would keep QI at that fixed value and solve for the value of PU. Again, keep PU at that fixed value and solve for the value of QI. And keep repeating the process until the values of PU and QI don't change anymore each time you solve the equation. Just like all the other techniques we have seen until now, alternating least squares is available in a pre-built library, so you don't have to actually implement this particular logic that we just talked about by yourself. All you need to do is to fill your algorithm in by telling the number of hidden factors that you want to look for. Based on the number of hidden factors that is specified by the user, the algorithm will find that many number of factors to represent each user and each product. The more the number of factors that you use, the more sophisticated or complex the model that you will have built for these recommendations is going to be. But in general, in machine learning, there is an underlying principle that you would like to find as simple a model as possible. If you have a choice between a model with 10 factors or 1, 000 factors, but the error between the two is only incrementally different, then you would go with the model which has 10 factors. In order to take this into account, we add something called a regularization term to our error function or the objective function that we are trying to minimize with alternating least squares. This regularization term has a factor called lambda, which tells us how important the regularization term is to our error function, and this lambda is one of the parameters that you will tweak when you are implementing ALS. So when you implement ALS in

Python, the number of factors and lambda are parameters that you can experiment with to make your model perform better.

Implementing ALS to Find Movie Recommendations

Let's implement movie recommendations using a public data set. This data set is called MovieLens. The link to download this data set is available onscreen right now. The data set has been created and is maintained by a group called Grouplens this is a research group at the University of Minnesota. This data set has a file called U. data, and this is a snapshot of the data from that file. The file has four columns: user, movie, rating, and timestamp when that rating was given. Let's see a demo in Python that uses this data set and implements the alternating least squares algorithm for movie recommendations. We'll be using a module called implicit for this particular algorithm, so make sure you install this module and import it before you start. Read the data file, which contains the user ID, item ID, and the rating. We can ignore the last column, while reading this file, which is just the timestamp at which the rating was given. If you just look at a sample of rows from this file, you will see that we have three columns with the user ID, item ID, and rating. This bit of code here converts that data frame, which has three columns, into a matrix, where the rows represent users, columns represent items or movies, and the cells represent ratings. Now, it's as simple as calling the alternating least squares method from the implicit module, and giving this matrix to that module along with the two parameters for alternating least squares. The result of this step will be that the user/item rating matrix is broken down into two matrices: user factors and item factors. Alternating least squares requires two parameters. The number of factors that it needs to find and a regularization parameter. Now that we have the user factor and item factors matrix, we can take one row from the user factors matrix, and that will represent one user. If you take the dot product of that vector with all the columns in item factors, you will get the predicted ratings for all movies by this user. Then, all you need to do is to sort these ratings in descending order and pick the top three. The top three rated movies will be the recommendations for this user. That brings us to the end of this module on recommendations. Let's quickly summarize what we've learned. We've understood the role of personalized recommendations. We are able to predict user product ratings using a collaborative filtering algorithm called latent factor analysis. Latent factor analysis uses a technique called: alternating least squares. We've implemented alternating least squares in Python. And used that to find movie recommendations.

Clustering Large Data Sets into Meaningful Groups

Understanding the Clustering Setup

Clustering is one of the other big categories of machine learning problems. And it is somewhat different from all the other categories of machine learning problems that we have seen until now. In this module, we learn how to spot applications of clustering in the real world. We'll be able to see the difference between classification and clustering problems, and we'll be able to understand how the K-Means Clustering Algorithm can be used to perform document clustering. We've talked a little bit about clustering at the beginning of this class. Given any large set of objects, we just want to group the items in that large set such that the objects which are similar are in the same group. Let's understand this by taking an example. Let's say we want to understand how users behave at the social network, and we want to articulate what are those different patterns that drive user behavior. We're going to perform clustering, and the objective of clustering is to take this large group of users and divide them into clusters or smaller groups. The way this is done is such that users in the same group must be similar to each other, and users in different groups are dissimilar to each other. So you are trying to maximize intracluster similarity and minimize intercluster similarity. The idea here is to find these clusters and study them and understand them and use that understanding to maximize the utility from each cluster. Whether is by creating specific promotions or introducing product features which are specific to each cluster. We're clear on what we want to do. Let's see how it is actually done. Features as we know are numeric attributes, and users can be presented using attributes like age, location, frequency of usage, or some preference called for each topic or keyword that might be used on the social network. If you represent each user using a tuple of such numbers, that tuple represents a point in some N-Dimensional space. So let's see here are all our users represented as points in an N-Dimensional space. The clustering algorithm will try to find _____ groups within this complete set of users. Here's one way to take the set of users and divide the groups. In this case the algorithm finds five clusters of users, each of them slightly different. Here's another way to do the same thing. Here we have two clusters of users, one blue and one red. Each cluster has some common attributes within them. We don't know what those are yet. Usually it's quite clear what's happening here. The similarity of users is based on the nearness or the distance between users. The closer to users are, the more similar those two users are. We are minimizing the average distance between users within a cluster and maximizing the average distance between users in different clusters. Now this

nearness or distance is in some abstract N-Dimensional space. But in real life it might have some real meaning. It might translate to. So here's the typical clustering setup. You would start with a data set which has a large group of items that needs to be broken down into smaller groups. Then you would represent each item in that large set using numeric attributes. Feed this data to a clustering algorithm, and the algorithm will give you all the items grouped into different clusters. Usually when you perform clustering, there are certain kinds of patterns that you are actually looking for. You should choose attributes relevant to the groups or patterns that you're looking for. Let's say you want the group users based on the similarity of their usage patterns. You want to study if there are different groups of users who have a certain routine when they log in to your social network. Then you could use attributes like the frequency of morning login, the frequency of evening login and so on. And the time that they spend per session or the time they spend per session at different times in the day. On the other hand, if you want to group users based on their interests, you could take a list of hundred topics and represent each users based on their likes and shares for each of those topics. Once you have represented your data using features, you can use a clustering algorithm to help you actually find the clusters. There are several different types of clustering algorithms available. Here are a few examples. Are all examples of clustering techniques that you can use to find clusters in large data sets.

Contrasting Clustering and Classification

We've talked quite a bit about classification before. Classification involves taking an object and classifying it into one of a set of predefined categories. Clustering also involves grouping data into a set of categories. On the face of it, it seems like both of these are actually doing the same thing, but that is not true. Classification is taking data and putting it in pre-defined categories. And in clustering, the set of categories that you want to group the data into is not known beforehand. This is a key difference, and understanding that difference will help you clearly differentiate between when it's appropriate to use classification, and when it's appropriate to use clustering. So let's dig a little deeper into this difference. In classification you would start with one instance, one object to be classified, then you would classify it into a pre-defined category. You have a set of labels. You will assign one of those labels to this instance. You would do this based on training data which has already been classified. For instance, in sentiment analysis you would classify one common as positive or negative, and you would do this based on a set of training data which has already been classified into positive and negative comments. Classification problems take a typical form asking which category a particular object belongs to. So, the problem statement in classification deals with one instance at a time. On the other hand when

you come to clustering, you're actually taking a large number of instances and dividing them into groups. So the problem statement itself takes a large number of instances into account. The groups that you are going to divide these instances into are completely unknown beforehand. You don't know before the clusters are formed what to call those clusters because you won't know until then what will be the common characteristics inside each cluster. If you look at clustering problem statements, you have statements like what kind of segments or groups can we find in a group of users? How can we divide the set of articles such that some articles have the same thing? Once again, we don't know the user groups ahead of time. We don't know the themes of the articles ahead of time. A typical classification setup involves four steps that we have seen already. The problem statement has to be defined, then you would represent your data in the form of numerical attributes called features. This is done both for training data that is already classified and for test data that has to be classified in the future. You would take your training data and feed it to a classification algorithm to train a model. And then take new instances that need to be classified or test data and pass it through the classifier to actually be classified. So we have four steps in the classification setup. Now let's move to the clustering setup. In the clustering setup, you would start with your problem statement again, which is the data set that needs to be clustered. Then you would represent points in that data set using features. So until here we are pretty similar to what's happening in classification, but we do not have a training step in clustering. You would directly feed your data to a clustering algorithm to find the final clusters, and you do not have any training step. Finally, let's summarize the differences between classification and clustering. In classification, you would take one instance and assign a label based on already labeled items. Since classification involves a training step, it is an example of supervised learning. Clustering on the other hand is an example of unsupervised learning. This is the first example of unsupervised learning that we've come across. All the other problem categories that we saw before recommendations, regression, classification, are all actually examples of supervised learning because they all have a explicit training phase. Clustering is the only example that we've seen so far where we do not have a training phase. In classification, this is data which is already classified. In regression, this is data for which the both the independent and dependent variable are known. In recommendations, it's the user product ratings that are available from which we can derive ratings for other products. In unsupervised learning, we just take a large set of data, this also happens to be called training data, but for this training data the output we are seeking is not known. In clustering, we don't know what groups should be assigned beforehand. In general, we use supervised learning techniques when we know what the output that we're looking for actually is. And we use unsupervised learning when we are looking for interesting patterns or exploring a data set, just trying to understand it, and not to

build a system for which a given input should lead to a specific output. We have clearing contrasted classification and clustering, but actually these two techniques sometimes go hand in hand. They're very complimentary to each other in some ways. You might imagine a situation where you have a set of articles, and you want to divide these articles into clusters based on their content. The output of clustering is the articles grouped based on their themes, and if you study the clusters you might realize that cluster one represents theme A, cluster two represents theme B and so on. Then, you have a set of themes that you know generally represent all the articles. Now when you see any new article, you would expect that it belongs to one of these themes, and you could figure out which theme it belongs to by using classification. So you would ask the new article to classify it. The classifier will assign one of the themes from the themes that you discovered during clustering. In fact, the articles assigned to different clusters is becoming the training data for the classifier. In this way, clustering and classification can actually go hand in hand and help you classify articles.

Document Clustering with K-Means

Whether you have an article, a comment, a book, webpage, review, or a tweet all of these are pieces of text. Document is the general term we use for a piece of text, and one of the interesting problems that you might want to solve with clustering is to perform document clustering. That is given any set of documents, you would want to group them based on how similar they are in terms of content. Once you have studied the identified clusters, you might find that there are interesting themes in the clustering themselves. So, each cluster might represent one particular topic that the articles are dealing with, and the clusters represent the overall set of themes across all articles. Our objective in this video is to understand how we can perform K-Means Clustering on the set of documents. Let's first talk about the features that we'll need represent each document. Whenever you want to represent text using numeric attributes there are different techniques that you can use for this purpose. One of the popular techniques is term frequency representation where you take each text and represent it using the frequency of words which are present in the text. So here's an example, we have the complete universe of all the words present in any text in this set. Then we have a sentence, hello this is a test. That sentence is represented using this tuple of numbers. 111 for the words hello this is, and then the rest of the words are represented using zero because they are not present in the sentence. Then, we have one and one again to represent the frequency of the words a and test. So the term frequency representation is nothing but using the frequency of words or terms in the text to represent the text. There's one enhancement that we can make to this representation to make it more meaningful. Some words

in a sentence might characterize that sentence more than others. For instance, in this sentence the words house and New York are clearly words that differentiate the sentence from other sentences and the other words, which are too common, do not really add much information to differentiate the sentence from other sentences. Words which are rarer in general are more useful in clearly differentiating a document from other documents. So in our representation we would like to upgrade words which are good rarely. Similarly, we want to downgrade words which are very common and don't do much to differentiate the document. This representation is called Term Frequency-Inverse Document Frequency. We weight the term frequencies to take into account the rarity of a word in the entire set of documents. The weight we apply to the term frequency is the inverse of the number of documents the word appears in in the overall purpose. So this representation is called TF-IDF or Term Frequency-Inverse Document Frequency. Once you have used the TF-IDF representation to represent your documents, you can feed that data to a clustering algorithm. The algorithm that we want to talk about is the K-Means Clustering Algorithm. This algorithm will divide your data into K clusters where K is specified by the user. We have our documents which are represented using TF-IDF. A document which is represented using a tuple of N numbers is actually a point in an N-Dimensional hypercube. Whenever you see a tuple of numbers, it's possible to imagine it as a point in some N-Dimensional space. So here we've got all our documents represented as points in an N-Dimension space We start by initializing a set of points as the K means or the K centroids of the clusters that we want to find. K here is specified by the user. So if the user specifies that we want to find five clusters, then five means will be initialized. The initial means can be anything, but there are several techniques to help find the right set of means in the first iteration so that the algorithm will reach conversions faster. However, that's not really in scope for our discussion today. So we'll just talk about what happens once you initialize the first set of means. Each document, or point, is assigned to the cluster belonging to the nearest mean. So you'll find the nearest mean for each point and assign it to that particular cluster. Once you've done that, take each cluster and find a new mean or centroid for that cluster. So from the initial points, the cluster centers move to the new cluster centers. Then, once again, we assign the points to the nearest cluster. We keep repeating steps two and three where we assign the points to a cluster and then find the new means until the means don't change anymore. That point where the means don't change anymore is called convergence, and finally we'll have the five clusters that we want to divide the documents into.

Implementing K-Means Clustering

Let's take a bunch of reviews and actually perform K-Means Clustering on that set of reviews. The data set that we're going to use is from the University of California Machine Learning Repository. This is the same sentiment labeled sentences data set that we already looked at for the _____ based sentiment analysis problem. This data set contains reviews from Amazon, IMDB, and Yelp. We'll take the IMDB movie reviews data set and see if there are some movie reviews that can be clustered into different groups and some themes that might emerge from those reviews. If you recall each line in this data set actually has two values. The review itself and the label saying whether this is a positive review or a negative review. This label is not required for us because we won't actually be performing any training step. We just need to take the reviews as is and feed it to a clustering algorithm. We'll do that in _____ using K-Means Clustering from the sidekick learn module. We start off by reading the data from the text file. This step is exactly the same as what we did in the _____ base case, except that we're just reading one file now. That is the IMDB _____. Then we split up the lines into reviews and labels taking care to get rid of any corrupt data. And we just pick the first element in each row which is the review itself. We need to take the set of reviews we've got and represent them using features. We'll create the features using the TF-IDF _____ from sidekick learn. This will take our reviews data set and convert each review to its TF-IDF representation. Each review will be represented with a tuple of numbers which are word frequencies multiplied by the inverse document frequencies. We instantiate a TF-IDF _____ with some basic parameters. The max TF and min TF are parameters that you can experiment with to see the changes and the results. The stop words variable is used to get rid of words such as the, a, an, and so on, which don't really add information in terms of the document's theme. The TF-IDF _____ will take care of removing the stop words if you just specify the language. Then we use the fit transform method to actually convert our training documents into the TF-IDF form. Now all that's left is to actually pass on this data to a clustering algorithm which we do by instantiating a K-Means clustering object and passing it the training documents. Just specify that the number of clusters to be found is three. We've also specified the max number of iterations that K-Means should go through in case it does not converge. Then it should stop before hundred iterations. At the end of the step, every document has been assigned to one cluster. The K-Means algorithm will assign a cluster number, either zero, one, or two, to each document. And this cluster number can be accessed using a property called labels. _____ a loop here that will quickly print out three items that have the label zero. So here are three documents from cluster zero. Similarly, if you just change the label to one, you'll get three documents from cluster one. And if you use the label two, you will get the documents from cluster two. At this point, you will need to actually do some deep exploration of what are documents inside each cluster to actually understand what are the themes that these clusters present. For instance, it

might turn out that one cluster contains reviews which comment upon the technical aspects of the film, and another cluster might contain reviews which comment upon the story of the film. That brings us to the end of this module on clustering. We've learned how to spot applications of clustering and how to differentiate between classification and clustering. We've also seen that classification and clustering can sometimes be used in tandem to solve some problems. We've understood how the K-Means Clustering Algorithm works, and we've applied it to actually cluster a set of reviews.

Wrapping up and Next Steps

Surveying Machine Learning Techniques

We have come a long way on this journey developing our thought process of understanding and using Machine Learning algorithms. Let's take a little bit of time to wrap our heads around everything that we have learned. We'll recall when it's appropriate to use Machine Learning, how to Recognize and differentiate between the important types of Machine Learning Problems that we have learned about and, finally, we'll also understand what are the next steps for someone who wants to go deeper into Machine Learning. We'll look ahead into the different areas in which you can dig deeper to become an expert practitioner of Machine Learning. We'll start off this journey by talking about this example of an alien who was observing happenings on Earth. And we said that the best way for this alien to understand what is happening on Earth is by gaining experience, with the help of a human, guiding them about what's happening on Earth. This is very similar to what's happening in Machine Learning because a machine is actually like an alien, they don't really understand the phenomena around them like humans do. In Machine Learning we are actually trying to build a computer program or system that, given enough experience, can learn and understand what is going on. And the experience, or guidance, that it is getting is in the form of data. In some ways Machine Learning is trying to make machines more intelligent, because it is helping them to solve problems like Spam Detection, Recommendations, and so on, which are traditionally best performed by human beings. For instance, human beings are better at identifying what is Spam in their inbox. And sometimes, sales people are really good at identifying recommendations for products. But by allowing a Machine to learn patterns and relationships from data the machine can solve these problems as well, or even better than, a

human. While this sounds like a hefty endeavor it can actually be broken down into three simple steps. You start by identifying which type of Machine Learning problem you are trying to solve. Then you represent your data sect in terms of numeric attributes, which are called features. And finally, apply an algorithm that works for that category of Machine Learning Problems. Let's recall what we can do in each of these steps. You start by identifying whether your problem is a Classification, Regression, Clustering, or Recommendations problem. This is crucial because all the choices you make following this step will completely depend upon the type of problem that you are trying to solve. You identify the problem based on the problem statement. If the problem statement is to classify something into a pre-defined category, then it's a Classification problem. If the problem statement involves quantifying the relationship between two sets of variables or computing a continuous value, then you would choose Regression. If the problem statement involves studying user behavior and then recommending products that a user might like, whatever the products might be, movies or news articles, or products on Amazon. Then you are solving a Recommendations problem. If the problem statement involves taking a large data set and finding meaningful groups or hidden patterns or themes, then you should use Clustering. Once you've identified the type of problem you are trying to solve, then you would go ahead and Represent your Data using numeric attributes. In Classification you would use the attributes of the instance you want to classify, or the object that you want to classify as features. In Regression you would use variables that might be related to or influence the output variable. These variables may or may not be specific to one object or once instance. Recommendation Problems, most often, use product ratings by users or preference scores by users for products. These might be explicit ratings collected in some way, or implicit ratings in the form of user clicks or purchases. In class today you would take the data set that is to be grouped, and use attributes based on the insights that you're looking for. The grouping will then be done based on similarities within those attributes. Once you have represented your data, now you are ready to apply an algorithm to this data. Either with two steps, with a training and test step. Or in just one step, as in the case of clustering. Then depending upon the type of problem that you are trying to solve, there are different algorithms that you can use. For instance, for Classification, you could use algorithms like Naive Bayes or Support Vector Machines that we have already seen. Or there are quite a few other algorithms like decision trees and random forests, which come under Tree based models. And Nearest Neighbors of Logistic Regression. In the case of Regression you could use algorithms like Linear Regression or Non-Linear Regression. Depending upon the form of the relationship between variables that you are looking for. There are actually several interesting algorithms that help you find recommendations. We've concentrated in this course on one type of algorithm called Collaborative Filtering algorithms. Under Collaborative Filtering we have looked

at the Alternating Least Squares algorithm, but there are other Collaborative Filtering algorithms as well, such as the Nearest Neighbor Model. Association Rules and Content Based Filtering are some other types of Recommendation algorithms that we have not talked about in this course, but might be interesting to look at. For Clustering we have learned about the K-Means Clustering algorithm which is a pretty popular algorithm for Clustering. However, this may not always be appropriate, and there are other algorithms for Clustering as well. Such as Hierarchical, Density Based, and Distribution Based Clustering. So depending upon the type of Machine Learning problem that you are trying to solve, there are several different choices for the algorithm that you can use.

Looking Ahead

Let's talk about what's next for someone who is interested in Machine Learning. Here are the three steps in a typical Machine Learning Workflow. In this course we have focused significantly on the first step which is picking the type of problem. Our focus has been on identifying when it's appropriate to use Machine Learning, and then identifying the type of Machine Learning Problem that we are trying to solve. Coming to the step involving representation of data. While we've talked about a few specific examples and how to Represent the data in those cases, there are quite a few challenges involved in this step. And solving those challenges requires some specific skills. Let's talk about each of these challenges and what kind of skills you can double up to solve each of them. Data is at the heart of Machine Learning. And most of the data which is generally available requires heavy duty pre-processing before it's ready to be used. The data might have missing values, and identifying and accounting for that will become an important skill. Similarly, recognizing and fixing corrupt data can mean the difference between a good model and a meaningless model. Developing programming skills in platforms such as Python, Spark, or R, which are particularly friendly for data exploration and analysis, can make the step of Data Munging relatively straight-forward to achieve. It's important not to underestimate the importance of this step. And many Machine Learning practitioners spend a significant amount of time in Data Munging to get their data ready to be used. Feature Extraction is another step which we have talked about a little in this course. But which is quite last in its school. Data that we get is often Unstructured, Semantically complex, and it might be in different forms such as Images, Video, or Text. Depending upon the type of data that you are dealing with there are quite a few techniques available to extract features from that data. Natural Language Processing is an area which helps with extracting features from text that can be used for Machine Learning. Image and Video Processing techniques are required in order to be able to extract relevant features from

images or videos for Machine Learning algorithms. A lot of the times when you are processing images or text you would end up with 100's or 1000's or features. And it's difficult to realize what is actually relevant to your model and what is not. If you try to use all of these features that would actually lead to an explosion in the Computing Complexity. And using techniques like Principal Components Analysis or Feature Selection helps you deal with the so-called course of Dimensionality. Finally, there is the field of Feature Engineering. This is a somewhat informal field - somewhat of a black art because only very experienced Machine Learning practitioners are very good at it. And it's not formally explained in any textbooks. This basically involves taking a set of features that you already have, and combining them or using them in some way to construct more relevant features or features that will make your Machine Learning algorithm perform better. All of these are different examples of areas that you can dig deeper to become better at representing data using features. A lot of good Machine Learning practice goes into getting each of these steps right. Finally we come to the third step in the Machine Learning Workflow which is Applying an Algorithm. Now we've talked about different types of algorithms for different Machine Learning Problem categories. But there's one important aspect of this that we haven't really covered, and that is Model Selection. Given the type of Machine Learning problem we actually have, sometimes, 10s of choices for the algorithm to solve that problem. Even if you choose one algorithm, within that there might be several parameters that you can tweak. We have seen this with K-Means clustering, for example, where it is up to the user to choose the number of clusters. Or in the case of statistic radiant decent the user might have to choose the number of integrations or the step size to be taken. A lot of the times the question here might be how do you make the right choice. And the answer to that is Model Selection. Model Selection techniques help you evaluate the performance of an algorithm and specific parameters, and choose the best model for your specific problem and data set. Hyper Parameter Tuning, Cross Validation, Ensembling. These are all different techniques which help in Model Selection, and are pretty interesting to explore and understand. That brings us to the end of this course on Machine Learning. We've understood when it's appropriate to use Machine Learning. We've understood how to recognize and differentiate between different types of Machine Learning Problems. And we've given you a sneak peek at the different areas that you might want to explore further. This is Swetha signing off for today. I hope you enjoyed this course and look forward to seeing you next time.

Course author

Swetha Kolalapudi

Swetha loves playing with data and crunching numbers to get cool insights. She is an alumnus of top schools like IIT Madras and IIM Ahmedabad. She was the first member of Flipkart's elite...

Course info

Level	Beginner
Rating	★★★★½ (264)
My rating	★★★★★
Duration	3h 8m
Released	27 Sep 2016

Share course