LECTURE TRANSCRIPTS

MIT Professional Education

Data Science: Data to Insights

Solve Complex Issues With Your Data

Contents

Welcome to the Course

<u>Welcome Remarks from the Course Co-Directors (Philippe Rigollet and Devavrat Shah)</u>

Hello everyone, welcome. My name is Philippe Rigollet. I am a professor with the Department of Mathematics and the Institute for Data, Systems, and Society at MIT. I am co-director of the course on data science, data to insights.

Hello, my name is Devavrat Shah. I'm a professor with the Department of Electrical Engineering and Computer Science and Institute for Data, Systems, and Society at MIT.

I'm the other co-director of the course. Before we get started, we would like to tell you a bit about why we are excited about this course.

The defining characteristic of our times is access to an abundance of data. Effectively, everything that we do is recorded. What we eat, what we watch, how we socialize, what we buy, and what our machines do.

It almost appears like the DNA of our lives is recorded. The systematic collection of data is meant to replaced targeted studies or surveys there are often done in much smaller scale and higher cost.

The access to this data presents us with tremendous opportunities. Extracting information from this data can help us make better decisions.

For example, federal government can make better policy decisions. Businesses can operate more efficiently and profitably. Public resource utilization can be made more efficient. Personalized health care and drug design could be feasible. Overall, quality of life of society will improve, and so on. The key to realizing these major opportunities lies within our ability to convert the available data into meaningful decisions.

Over the past few decades, we have built infrastructure that can store and process massive amounts of data. However, we're still lacking in the critical ability to seamlessly stitch various pieces of data together to make meaning predictions, and subsequently, impactful decisions. Given the endless opportunities that can be unlocked by addressing this challenge, I would say that this is one of the defining challenges of our times.

As an educational institute, we can play an important role in overcoming this important challenge, specifically through courses like this one. We hope to teach the science behind handling large amounts of data as well as extracting meaningful information out of it. The purpose of this course is to help you take steps towards becoming an excellent data scientist and statistician.

This course is taught by experts in statistics, machine learning, econometrics, and social sciences. This combined expertise results in the streamlined interdisciplinary curriculum that will help you understand the important data mysteries of our times.

The course is designed to have a modular structure. It has five key modules.

Each one of which is chosen based on its importance when dealing with data across different industries. The first module is about discovering structure in the data as well as obtaining insights from it. The tools and techniques developed in this module will allow one to take a peak at large and unstructured data.

It is naturally the first module as this is often the first step of data processing. This module is taught by professors Tamara Broderick, and Stefanie Jegelka.

Once we get better handle on what this data looks like and what information it contains, a natural question is whether we can use it to make predictions.

The need for prediction from data can arise in a variety of context, whether we are trying to predict future data or simply data that is hard or expensive to collect in future but available in the past. That general idea behind prediction is to analyze data in order to identify an input and output function relationship.

This is discussed in the second module, which focuses on regression and predictions. This module is taught by Professor Victor Chernozhukov.

The module on regression and prediction also covers confidence intervals, which provide error bars rather than a single prediction. In some sense, prediction and confidence intervals are complicated questions, because they require data to speak for itself.

In a much more directed setting where data is asked simpler yes/no question, we can give more accurate answers. This is the premise of hypothesis testing, anomaly detection, and classification. This topics are covered in the module in classification, hypothesis testing, and deep learning. A beautifully unified view of such diverse topics is presented by professors David Gamarnik, Jonathan Kelner, and Ankur Moitra.

Regression and classification are basic prediction tasks, and in some sense, the next module on recommendation systems introduces a more refined notion of prediction, indeed. Implicitly, the models behind regression and classification make a strong assumption. All the points in the data set have the same input and output function relationship.

It turns out that relaxing this assumption to have personalized prediction is key to a comparative recommendation system. In this module, Professor Philippe Rigollet and myself demonstrate why recommendation systems are now everywhere, give us some insight on what is required to build a good recommendation system by covering statistical modeling and algorithms.

In either regression, classification, or recommendation systems, we saw that correlations or interactions played an important role. Sometimes, data comes directly in the form of interactions, or we are directly interested in understanding interactions rather than making predictions. This is the case, for example, in social networks or gene regulatory networks.

Local interactions between basic entities in a network can give rise to a large scale network effect, such as the spread of information or diseases. How do we make use of network data to understand the behavior or functionality of the network? The fifth and final module provides a systematic overview of methods for analyzing large networks, determining important structure in such networks, and for inferring missing data in networks.

An emphasis is placed on graphical models, both as a powerful way to model networks processes and facilitate efficient statistical computation. This module is taught by Professor Guy Bresler and Professor Caroline Uhler.

All modules involve hands-on case studies that explain how to utilize various material and methods learnt in the course to practice.

Each module is divided into multiple segments, and each segment comes with a useful assessment that will help making sure that you are following the material well. There'll be plenty of online help available in case you have any questions, so do not be shy, and ask questions.

The course will prepare you to undertake the challenges you have in your respective profession and industry segment so that you can enable data-driven decision-making.

Our goal is to help you understand what kind of data you should collect and what kind of method and algorithms you should utilize to make predictions and decisions. We will point out various system infrastructures along the way as it makes sense. So at the end, you should be able to put together a team of engineers to build a data-driven system of your own.

Good luck with the class, and we will see you at the other end.

Module – 1: Making Sense of Unstructured Data (Stefanie Jegelka and Tamara Broderick)

1 Introduction

1.1 What is Unsupervised Learning, and why is it Challenging

Hi and welcome to this module on unsupervised learning. So what is unsupervised learning? In unsupervised learning you're trying to discover hidden patterns in data when we don't have any labels. We'll talk about what hidden patterns are and what labels are shortly. And we'll talk through a lot of real data examples.

But before that, let's introduce ourselves. I am Stefanie Jegelka. I am Tamara Broderick. Both Stefanie and I are professors at MIT, with the Department of Electrical Engineering and Computer Science. And also with the Center for Statistics at the Institute for Data Systems in Society. I work on machine learning methods for all sorts of combinatorial data, things like networks, subsets, or images. From theory to making them work on real data.

I work on machine learning methods for Bayesian and nonparametric Bayesian learning, that scale to very large data sets. I often work on methods for unsupervised learning. Stephanie and I will be your instructors for this module. So what is unsupervised learning again? And, in fact, is supervised learning a thing at all? This is a great observation.

First, let's step back to what learning even means here. In machine learning and statistics, we're typically trying to find hidden patterns in data. Ideally, we want these hidden patterns to help us in some way. For instance, to help us understand some scientific result, to improve our user experience, or to help us maximize profit in some investment. Supervised learning is when we learn from data but we have labels for all the data we've seen so far. Unsupervised learning is when we learn from data, but we don't have any labels. Some examples would be really helpful.

Absolutely, let's start by thinking about email. I don't know about you, but for me it's really hard to keep my inbox in check. I get so many emails everyday. And a big problem here is spam. In fact, it would be an even bigger problem if email providers like Gmail weren't so effective at keeping spam out of our inboxes. But how do they know whether a particular email is spam or not?

So that's our first example of a machine learning problem. Every machine learning problem has a data set, that is a collection of data points that help us learn. Here, our data set will be all the emails that are sent to some MIT professor over say, a month. Each data point will be a single email. Whenever I get an email, I can quickly tell whether it's spam. So I might hit a button to label any particular email as spam or not spam.

Now we can imagine that each of our data points has one of two labels, spam or not spam. In the future, I keep getting emails, but of course, I won't know in advance which label it should have, spam or not spam. So the machine learning problem is to predict whether a new label for a new email is spam or nor spam. That is, we want to predict the label of the next email. If our machine learning algorithm works, we can put all this spam in a separate folder and not have to worry about it. This spam problem is an example of supervised learning. And now, we can start to see why it's called supervised learning. You can imagine some teacher or supervisor telling you the label of each data point.

That is, whether each email is spam or not spam. And we often want to check how well our machine learning algorithm is doing too. Then a supervisor might be able to tell us whether the labels we predicted were correct. When I got a new email and I said it was spam using my algorithm, was I really correct about that or did I just throw a really important email into the trash?

Okay, so that's supervised learning. So what is then unsupervised learning? What do I learn here? Let's try another example of a machine learning problem. Now imagine I'm looking at my emails and I realize, man, I have way too many emails. It would actually be super helpful if I could read all the emails that are on the same topic at the same time. So, I might run a machine learning algorithm that groups together similar emails. It turns out I'm rushing around so I don't have the time label emails, spam or not spam or anything else. I don't even have the time to come up with topics. But after I run my machine learning algorithm, I find that there are natural groups of emails in my inbox.

There are all the emails about upcoming and past hiking trips. There are a bunch of emails about the machine learning class I taught last semester. And there are emails about a cool machine learning research project I'm currently working on. Now this is an example of a unsupervised learning problem, you didn't have any labels because you didn't have time to give any labels. And since no one else made labels for each email, there's no supervisor. In the previous supervised learning example, the machine learning algorithm had to find hidden patterns in the data that helped predict whether any email is spam or not spam. In this unsupervised example, the machine learning algorithm only has the context of the emails to find hidden patterns. There aren't any labels to help it out.

Unfortunately, my email client doesn't actually group my emails by topic yet. But you actually see this kind of thing all the time on other websites. For instance, have you looked at Google News recently? Google could just list all the news articles it finds and order them by say, the time they were published, but that would be overwhelming. And when you go to their site, you'd never be able to find out what the interesting news stories are. Instead, Google groups together all the articles that are about a similar story. That way you can see the major trends and then dive into lots of different articles on a single topic.

Yes, same thing with Facebook trending stories. If you're on Facebook, you'll see a list of big stories, but these are just groupings of a lot of individual articles and posts. You'll see the individual posts if you click on the big story. We're starting to see some of the benefits and challenges of unsupervised learning. Sometimes it's really expensive to get labels. I could always hire someone to come up with topics for my emails, and to label each email with some topic, but then I have to pay that person's salary. Plus it would take them quite some time to label everything. Humans take a long time to read emails, much longer than a computer does. That's the whole problem to start with. It's cheaper for me in terms of both time and money if a machine learning algorithm can sort my emails automatically.

On the other hand, it's often a lot harder to do unsupervised learning. The labels give us a lot of information. We can identify words that would tend to occur in spam emails and words that tend to occur in emails that aren't spam. When we don't know what kind of hidden patterns we are looking for, it can be much harder to find a pattern that's actually important. My algorithm may group together all the emails that come in before noon, and make a separate group for all the emails that come in after noon. Well, that's not very helpful to me. But since I didn't tell the algorithm what I wanted, well maybe I shouldn't be surprised.

Plus, it's easier to tell how well a supervised learning algorithm is performing. Just check what percentage of spam and not spam emails it's getting correct. But is there a really quantitative way to check whether my emails are getting a good grouping by topic? Or a way to check whether Google News is finding a good grouping of stories? These are hard problems.

In the next video we'll talk about some more supervised and unsupervised learning problems to make super clear what the difference is. And to get a sense of the many different flavors of unsupervised learning. Then, in the remainder of this module, we'll start to dig deep into how unsupervised learning really works in practice.

We'll see how to actually build an unsupervised learning algorithm to help find these patterns in your data. We'll talk about methods for evaluating these algorithms and we'll see some more real life examples of unsupervised learning in action.

1.2 Tools and Applications & How They Fit Together

Welcome back. In the last video we introduced two types of data analysis problems, supervised learning and unsupervised learning. We talked about why unsupervised learning is both particularly challenging and particularly important. And we discussed a data example. For supervised learning we imagined we had labels on a bunch of emails, whether they were spam or not spam. And we wanted to predict the label of future emails.

For unsupervised learning, we imagined that the emails were unlabeled, and we wanted to find hidden patterns in the data. For instance, we'd like to group the emails by topic. One interesting thing about these examples was that for both cases we had essentially the same data. It was really whether we had labels and what we wanted to do with the data that determine whether the problem was supervised or unsupervised.

This time we will go through some more examples to test your knowledge of supervised and unsupervised learning. And also to get a taste of the really wide range of machine learning problems that are out there.

Okay, so here's a data analysis problem. Suppose I've gone around and measured the concentration in the air of a certain type of pollution. I measured the amounts of pollution at different locations under different weather conditions and at different times of day and different days of the week. Now, I'd like to predict the air pollution concentration at a new location with known weather and known date and time. Take a moment to convince yourself whether this is supervised or unsupervised learning. Pause your video if you need to.

This is an example of supervised learning. The data point is the collection of information about your surroundings. What your location is, what the weather is, what date and time it is. The label is the air pollution concentration. And we observe that. You'd like to predict the label under new conditions. In the email span examples, our label was spam or not spam. That was a categorical label because the label was a category. There's no inherent order on those labels. In this pollution example the label is numerical and has an order, 2,000 particles per cubic centimeter is less than say 6,000 particles per cubic centimeter.

I've got another data analysis problem. When Frank Samson was a PhD student in the 60s he spent some months in the New England Monastery. He followed the novices training there and recorded their interactions with each other. We'd like to discover what their friendships or social groups among the novices monks are. Is this supervised or unsupervised learning? It's unsupervised learning. We don't have any labels that tell us which social groups exist or who belongs to which groups. We need to discover these patterns in the data in an unsupervised way.

But Samson's data is old but this actually a modern problem. We often want to detect communities or groups in large social networks like those on Facebook, or Twitter or even Fitbit. Only that today our networks are way bigger than what Samson could record in his notebook back in the 60s.

Okay, here's another one. NASA uses rocket boosters to launch spacecraft into orbit. It's expensive to build a whole new rocket booster for every slight tweak, and it's difficult to recreate real atmospheric conditions in a wind tunnel for testing these out. So NASA uses computer simulations. But it takes a long time to run the computer simulations for different rocket booster specifications. So, NASA would like to measure the simulated lift, say, for different rocket booster specifications, and predict the simulated lift under other specifications. Is this supervised learning or unsupervised learning?

It's supervised learning. The data point is the rocket booster specifications. Details that go into the rocket booster as built. The label is the simulated lift. We'd like to predict the simulated lift at new data points. It's interesting here that we're putting machine learning on top of other computer simulations. It's turtles all the way down. How about this one? I have some small cameras that I've installed to monitor birds' nests in New England. I don't have a lot of bandwidth to send video. So I'd like to compress the recorded videos before I send it to my laptop. Can I use supervised or unsupervised learning to help me out?

It sounds quite different from problems we've encountered before, but you can think of this as an unsupervised learning problem. Yes, if you consider the pixels at each time step as a data point, we can group together pixels that are similar in color and similar in time to achieve compression. We'll learn more about this in a later video.

Okay, one last example. Suppose I'm working at a computer science MOOC, one of these massive open online courses. And for a particular exercise, I'm having all of the students submit some code to implement a particular algorithm. I'd like to understand all the different types of errors they make, so that I can easily grade the work and formulate my next batch accordingly. Supervised, or unsupervised learning?

It's unsupervised. You can't anticipate all possible errors in advance, so it would be difficult to even try to label this data. You are trying to find hidden patterns in the types of responses students give based solely on their homework answers.

Exactly, and this might help me personalize the educational experience going forward. Students who made similar mistakes might profit from doing the same kinds of exercises next. Well at this point we've seen a really wide range of different supervised and unsupervised learning problems, but that's still only the top of the iceberg. In other modules you'll see further examples of supervised learning. But for the rest of this module you'll be diving deeper into unsupervised learning.

Today, we've become experts at identifying unsupervised learning problems, but in the remaining videos we'll see particular algorithms for unsupervised learning and how they work in practice. We'll develop a whole tool box for unsupervised learning.

Not only will you know an unsupervised learning problem when you see one, but you'll know how to solve it too.

1.1 Clustering

1.3 What is Clustering?

Hi and welcome to the sub-module on clustering. So far in this module on unsupervised learning, we've introduced the concept of an unsupervised learning problem in data analysis. We learned that unsupervised learning means finding hidden patterns in data, when we aren't given any labels. We saw in the previous videos, that there are a lot of different problems that we can solve with unsupervised learning.

But there's one special problem that stands out over decades now, as being the most popular version of unsupervised learning. And this problem is called clustering, what is clustering? In clustering, our goal is to group data according to similarity. Let's examine each of these three key words, group, data and similarity, in more detail to find out what clustering is really about, let's start with data.

Imagine that I'm leading an archaeological dig and every time someone on my team unearths an artifact, I have them record the position of the artifact. In particular, I can put down some marker near the archaeological site, and we can record how far north, and how far east each artifact was found relative to my marker.

This is the artifact location, the location for one artifact will be a data point for us. When I list all of the locations of the different artifacts, that's my data set. Maybe I have some colleagues who are historians, I can sell with them before my archaeological dig, and they tell me that three families lived in the area that we're exploring.

I bet you're already able to identify where the three families live, and which artifacts belong to which family. Let's slow down this process to understand what you're implicitly doing. First, there's a notion of similarity here. In this archeology example, we say that two artifacts, are similar or close, if they're close in physical distance.

In other words, if they're close in distance as the crow flies. For instance, these two artifacts are relatively close. The distance between them is the length of this line. And these two artifacts are relatively far. The distance between them is clearly much larger. Second, when you think of where the three families lived, you're grouping the data according to this distance.

You're signing each data point to one and only one family group. So that data points that are physically close tend to belong to the same group. That is you're grouping together similar data points. When we assign each data point to one and only one group like this, we call the groups clusters.

Here there are three clusters, one for each family. Clustering then is the unsupervised learning problem where you take your data and assign each data point to exactly one group or cluster, and of course we want these clusters to be meaningful. We want them to give us some interesting insight into our data set.

In this example, we imagine the clusters correspond to three different families who used to live in the area. The data points in any cluster are pottery shards or other items that we believe a single family discarded when they lived at this spot. Identifying the items that we think belong to one family, will let us perform further scientific or anthropological or historical analysis of the artifacts.

Now remember, in an earlier video we discussed the difference between supervised and unsupervised learning problems. There's a supervised learning problem that is somewhat similar to but not the same as clustering. That problem is called classification. In classification, like in supervised learning in general, we're given labels. For instance, consider our archeology example.

If we were running classification on this data we might not only know that there are three families, tut we might know their names as well. Maybe the names are Lannister, Stark and Targaryen. Classification is what we called our supervised learning problem, when the labels are categorical. Remember, this means that the labels have no order, like the family names I just mentioned.

Now let's go back to our archaeology example. If we had a classification problem here, we'd have been given a bunch of data points with labels. For instance, you might imagine each family actually made a lot of pots with a or other identifying mark. Then our goal would be to find labels for any remaining unlabeled artifacts.

That is, our goal would be to predict labels for new data points given the labels at all data points. By contrasting clustering, we don't know the families or their names. We might know that there are three families, but we don't know anything at all beyond that number, and none of our artifacts come with labels.

We have to discover how to group the artifacts from their locations alone. Labels may be hard to come by for various reasons. In the archaeology example, we just might not be lucky enough for families from thousands of years ago to have carefully labeled their pottery for our benefit.

On the plus side, clustering lets us find hidden groupings in data, even when we can't run classification. But just as we saw when we compared supervised and unsupervised learning, clustering can be a more difficult undertaking than classification, since we don't have the information contained in the labels. You might be thinking to yourself at this point, well, that archaeology problem was easy.

I can find the clusters myself, why did I need a computer to do it? But it's not always so easy. Here we have two dimensions to each data point. The distance north and the distance east. Often we're dealing with much higher dimensional data. In this case, we can't just make a picture of our data and look at it.

Also, in the archaeology example we could plot each data point and look at them all at once. Well you'll notice we had a lot less than 100 data points in this example. If we had billions of data points, it wouldn't be so easy to visualize what's going on.

Also in this case our data was a numerical. Each location was a number, that made it easy to make a picture and easy to see the distance between two points. But sometimes our data isn't numerical, it might take the form of words or pictures, or genomes or something else entire.

And again it isn't so easy to immediately see the hidden patterns in these types of data. But it will turn out that we can still get a lot of information from clustering. Finally, you might have hundreds or thousands of different data sets that need clustering. In this case you simply don't have the time to find and write down all the clusters by yourself.

It would be better to automate the process. In the remainder of this sub-module, we'll talk more about why clustering is both an important and a difficult problem. We'll see many more practical examples of clustering, we'll learn algorithms that let you find clusterings hidden in data in an automated way.

We'll talk why clustering has been so popular for so long, but we'll also talk about the limitations of clustering and why you might be interested in other types of unsupervised learning problems as well.

1.4 When to Use Clustering

Welcome back. In the last video we learned that clustering is a particular form of unsupervised learning. Generally, in unsupervised learning, we might be trying to find any hidden pattern in a set of data points without labels. In clustering, we want to find a latent grouping such that each data point belongs to exactly one group called a cluster.

Last time we saw a particular example of clustering in action. We thought about how we might use clustering to help us understand the arrangement of artifacts and archaeological dig. In this video, we'll go through more examples of clustering and see why and when you might want to use clustering in practice.

To start, sometimes we have a collection of data but we're not quite sure what to do with it yet. You might want to explore the data without a particular end goal in mind. Perhaps the data will suggest interesting avenues for further analysis. In this case, we say that we're performing exploratory data analysis.

For instance, I might imagine that I have some data on meetup.com users. Roughly each data point corresponds to a person. In order to apply clustering I have to define a notion of similarity too. I'll say similarity is the number of common interests between two people. Well, I can't actually feed a person or a user into my computer, so I need to be more precise about what constitutes a data point here.

Perhaps what I have actually collected for each person is a list or vector of whether that person is interested in each meetup.com group on the meetup website. Then it's easy to calculate the similarity between two users. It's the total number of groups, those two people are both interested in.

After I apply clustering, I might hope to find a groups of users with broadly similar interests. Maybe after I run clustering it turns out these users are interested in playing music and watching musical performances, these users are interested in outdoor sports, and these users are interested in hackathons.

What are some other times I might use clustering? Basically, I'm going to use clustering for problems that kind of look like classification but when I don't have any labels that I could use to divide up my population and the direction I'm interested in. For one, I might just not know the labels in advance.

Perhaps I have a corpus of documents. Like I have all the documents on Wikipedia or all the past articles from Wired Magazine or all the user reviews on Yelp. I'd like to find out what topics or themes are represented in these documents. In general, I don't know the possible topics in advance.

I might guess at some topics but I can't really be sure in advanced that I've got them all. Or topics I think of might not be the ones that would really appear in the corpus. Ideally I'd like to discover the topics. In this case we can apply clustering.

You might say each data point corresponds to a word and we can measure the similarity of two words by counting how many documents they both appear in. Finally, we can group together similar words to form our cluster. Okay, but we can't actually calculate how many documents two words about to occur in just by looking at the letters of those words, so we need to be more precise about what constitutes a data point.

Here, the data point that corresponds to any particular word could be a list or vector of whether this word occurs in each document. Yes, or no? Then we can actually calculate the overall similarity of two words. A student in my class last semester, Julia, looked at research abstracts from professors like Stephanie and me here at MIT, in the Department of Electrical Engineering and Computer Science.

She found the following words that tend to co-occur. Words like temperature, graphene, devices and magnetic, go together. Words like algorithm, model and data, go together. And words like quantum, state, channel and energy, go together. It turns out that the first set of words have to do with fabrication or manufacturing.

The second set of words are related to data science and the last set of words are related to physics. But don't worry if you wouldn't have thought of these topics on your own, here is the really neat thing. The student didn't use any advanced knowledge about what MIT professors are working on.

The different topics in the data came out in an automated way. This is just one of many applications where we might not know the labels of our clusters in advance. Sometimes, we don't have labels because it's expensive to label data. For instance, it might take a while for a human to label each data point, and the labels might be changing too quickly to keep up.

Take the example of Google News we mentioned in a previous video. We might want to put news documents about similar topics in a cluster. But the topics in the news are changing everyday. For instance, just hours after a new Game of Thrones episode comes out, there are a great many articles about that episode.

We'd like to detect all those articles and group them together. And we'd like to do this even when some surprise world news event happens that we didn't anticipate in advance. In this case, our data point might be a list of which topics occur in a given document. And our similarity measure between two documents might be a count of how many topics are shared by the two documents or how similar the topic proportions are between the two documents.

Alternatively, it might be expensive to label data, simply because there is way too much of it. For instance, there are ton of different images sharing services right now. And as a result there's a wealth of images online. Consider some recent news articles, about the discovery of a new species of bird.

These birds were found to live in parts of India and China. And this picture of the bird appeared in the news articles about it. We might want to automatically break up the image into the pixels corresponding to the bird, the pixels corresponding to the hand, and the pixels corresponding to the wall.

In general, it's time intensive to label a substantial number of pictures and pixels within them. We might use clustering instead. In this case, each data point corresponds to a single pixel, and our measure of similarity, might be how similar the pixel is to another pixel, in both color, and location.

Equivalently, we might say our measure of dissimilarity is some notion of distance in color between pixels plus distance in location of the pixels in the image. What does our data point need to tell us to calculate this similarity? Our data point for each pixel could consist of the color of the pixel, followed by the location of the pixel.

In this video, we've seen a lot of different reasons to use clustering. We've also seen a lot of different types of data where you might apply clustering. In the next videos, we'll learn about the most popular algorithm for clustering or probably for any type of unsupervised learning.

1.5 K-means Preliminaries

Welcome back, in the last two videos, we got a sense of what clustering is, as a particular form of unsupervised learning problem. Next, we're going to learn what is not only the most popular algorithm for clustering, but quite possibly the most popular algorithm within unsupervised learning. This algorithm is called the k-means algorithm.

Why has this particular algorithm been so popular for so long? First of all, it's fast, not only are the steps of the algorithm simple and quick to run, but it turns out the algorithm can easily take advantage of parallel computing. By running calculations simultaneously across multiple cores or processors, we can get even further speed ups and running time.

And finally, it's fast in programmer time. We'll see that the algorithm is straight forward to understand, and to code, and doesn't require the large number of parameter tweaks some other algorithms need. It's easy to dismiss this last advantage, but when you need results on a timeline, it's no small consideration.

Now, before we can learn the k-means algorithm, we need to understand the set-up and assumptions that go into the k-means clustering problem. Any method, in machine learning or statistics, will have assumptions built into it. To use any method effectively in practice, you have to know what those assumptions are.

Our first assumption, in this case, is that we can express any data point as a list, or vector, of continuous values. For instance, recall our archaeology example, we can write the location of an artifact as a list, or vector, with two elements. Some distance, say in meters east of a marker near our site, and some distance north of the marker, also in meters.

This data point is 1.5 meters east, and 4.3 meters north of the marker. So our data set is a list of all these vectors. We have one vector for each data point. Here, we suppose that we have capital N data points. In general, the information we collect for any data point, doesn't have to be a distance relative to some marker in an archaeology dig.

Usually we'll just call each component of the vector a feature. In this example we have two features, one for east and one for north. But more generally, we might have any finite number of features. Often we'll call the number of features d, where you might think of d as standing for dimension.

Now, recall that clustering is grouping data according to similarity. We've defined what a data point is for the k-means clustering problem, now we need to define what similarity means in the k-means clustering problem. And remember, last time we talked briefly about how it's equivalent to define dissimilarity for two data points, instead of similarity.

And that's what we'll do here, we'll say that the dissimilarity between two data points, in the archaeology example, is the distance as the crow flies. That is, the distance between these two points is the length of the line connecting them. This is sometimes called the Euclidean distance. One pair of data points that is farther in Euclidean distance than another pair, will also be farther away in the square of the Euclidean distance.

In this case, it will turn out we get the same result from k-means clustering, If we define dissimilarity as squared Euclidean distance, and this is straightforward to calculate. The square of the distance between two data points, is the square of the distance between them in one direction, say east, plus the square of the distance between them in another direction, say north.

Instead of writing out both square differences, a convenient way to write this sum is using a summation symbol. This notation is like a for loop from computer science. Let's suppose capital D is two for the moment, since we have two directions. Then this notation says, for each index, little d, from 1 to capital D, add the distance in the little d direction.

When capital D is 2, there are 2 terms in the sum. But remember, more generally, we might have more than 2 features and this notation let's us have as many as we want. Okay, we've got our data, we defined our dissimilarity, finally we need to group the data.

What does this mean? What is the output we expect from our algorithm? Well, the reason k-means is called k-means, is that we say upfront that we expect some number, k, of clusters. In the archaeology example, we chose k equal to 3. We'll talk later about what happens when you don't know k in advance, but the k-means clustering problem assumes you do know k.

Now, for each of these k clusters, we plan to get out a description of the cluster. In particular, we wanna know roughly what the cluster looks like, and which data points belong to it. To be more precise, to get a sense of what each cluster looks like, we'll get a cluster center for each cluster.

Let's call the cluster center for the little k cluster, mu sub little k. And then we want to know which data points are assigned to each cluster. So, let capital S sub little k, be the set of data points assigned to cluster little k. Remember that in clustering, every data point has to be assigned to exactly one cluster.

In this video, we've discussed the set-up and assumptions that go into the k-means clustering problem. Note, that the k-means clustering problem is more specific than clustering in general. We assumed that the data points can be expressed as continuous numbers. That's not really true if we have a yes/no vector, like in some of our examples from last time.

Even if we encode yes as one, and no as zero, we can't get any other values than those two in our vector. We also assumed a particular form for our dissimilarity. There are lots of other forms we might use, and we'll discuss some of them later. And finally, we assume that there are exactly k clusters where k is known in advance, that isn't always true in applications.

But now that we've got this setup in hand, in the next video, we'll see the k-means clustering algorithm in action.

1.6 The K-means Algorithm

Welcome back. In the last video we set up the k-means clustering problem as a particular subset of general clustering problems. In particular, we assumed that each data point is a vector of continuous values. And we're using squared Euclidean distance as our dissimilarity measure between two data points.

In this video we'll finally see exactly what defines the k-means clustering problem, and we'll see how the k-means algorithm is a particular algorithm one might use to solve this problem. Remember, last time we saw that the output of the k-means clustering problem should be a set of cluster centers, mu sub one to mu sub capital K, and a set of assignments of data points to clusters.

That is, capital S sub little k is the set of data points assigned to the little k cluster. The k-means clustering problem tells us exactly how we should choose the cluster centers and the assignments of data points to clusters. As we've said before, we want to group data points according to similarity.

So the broad idea will be that we want to minimize dissimilarity within each cluster. We've defined dissimilarity between two data points. What's the global dissimilarity? Well first we'll say that the dissimilarity between any data point and a cluster center is the distance from the data point to the cluster center.

We're still using square Euclidean distance here. The dissimilarity within a cluster will be the sum over the distances between data points in that cluster and the cluster center. And finally the global dissimilarity will be the sum over the cluster dissimilarities. To recap, we calculate the global dissimilarity by iterating over each cluster, over each data point within the cluster, and over each feature of the data point.

So now we can finally define the k-means clustering problem precisely. The k-means clustering problem is to minimize this global dissimilarity. We will typically call this quantity the k-means objective function. We will want to minimize it by choosing a set of k-cluster centers and assignments of data points to clusters.

Remember, we’re still assuming that we know the value of k. Okay, so that’s the problem, but how do we solve it? As with most problems in machine learning there’s no single silver bullet that will always get us the best answer. But in this case there is an algorithm that seems to perform very well in practice, and it’s called the k-means algorithm or sometimes, Lloyd’s algorithm.

Let's start by covering the algorithm in broad strokes and then we'll get into the details. We don't know the cluster centers in advance. So we start by initializing them to some values. Then we alternate between two steps. One, we assign each data point to the cluster with the closest cluster center, and two, we update the cluster center to be the mean of all the data points in its cluster.

We iterate these last two steps until we can't make anymore changes. That is, we repeat these steps until we've converged. Now let's talk through these steps in a little more detail. First, we need to initialize the cluster centers. You don't want to choose cluster centers that are far from all of your data.

One option is to draw the cluster centers uniformly at random from the existing data points. There are other, more sophisticated initializations you might try as well, such as K-Means++. You might also see these in popular statistical software. However we choose to initialize the cluster centers, once we have some values for the centers we can start iterating the main steps of the algorithm.

Recall we repeat these two steps until nothing changes. That might mean we check that the assignments of data points to clusters don't change between one iteration and the next or we might check with the k-means objective function that global dissimilarity doesn't change between one iteration and the next.

Now let's focus on the two steps within each iteration. First, we assign each data point to the cluster with the closest center, that is, for each data point we compute the distance to all k cluster centers. Suppose little k has the closest center. Then we assign this data point to belong to cluster little k.

Note, that this calculation can be done completely separately for every data point. Therefore, it's extremely easy to divide up the data points across cores or processors, and perform these calculations separately for speed ups and running time. This is called an embarrassingly parallel computation. Finally once we have the assignments of data points of clusters, we recalculate the cluster centers, in particular we visit each cluster in turn.

For the little k cluster we collect all the data points in this cluster and compute their mean. That is, for each feature we sum up the values of the data points in this feature and divide by the total number of data points in the cluster. The result is another vector with capital D features, and this is the new cluster center value.

Let's see a quick animation of the k-means algorithm and practice. We start with some initialization. For instance, as we said before we might choose three of our data points at random to be the first cluster centers. Then we iterate between assigning data points to clusters and choosing the cluster centers.

We, again, assign data points to clusters and calculate the new cluster centers. And again, and again, when we've reached this stationary point, we stop. A first and immediate question to ask is, does this algorithm always stop in finite time? Luckily for us, the answer is yes, always. You might try to convince yourself why this is true after finishing this video.

We'll leave a more careful evaluation of the algorithm output until next time. In this video, we've defined what the K-Means Clustering Problem is, and we've developed the K-Means Algorithm as a way to solve it. In the next video, we'll talk about how to know whether the results you're getting out of this algorithm are any good.

1.7 How to Evaluate Clustering

Welcome back. In the previous video, we define the K-Means clustering problem and describe an algorithm, that you might use to get a solution to the K-Means clustering problem. This algorithm is often called, the K-Means Algorithm or Lloyd's Algorithm. Now that we have a clustering algorithm in hand, we have to think about how to evaluate its output.

Our discussion won't be specific to the K-Means Algorithm, but that certainly one of many cases where we want to know how good our output is. Before we talk about how to evaluate a clustering of data, let's think back to the supervised learning problem of classification. In this case, we're given a set of data with labels.

And we want to predict the labels for some new data. For Classification, evaluation is straightforward if we have enough data. We typically set aside some data where we know the labels. The data points we feed to the algorithm, together with their labels, will typically be call the Training Data.

The data points we set aside for evaluation purposes are often called the Test Data. We use our Classification Algorithm to predict the labels on the Test Data and compare to the true labels. Then, we have an absolute universal scale. What percentage of the test data points did the algorithm get right?

We can easily compare other algorithms. A better algorithm gets more labels right. But we can also see, whether an algorithm on its own is good or not. Did the algorithm get a suitable proportion of the labels correct? Sometimes in clustering, one may have access to a ground truth clustering of the data.

In the archeological example, perhaps in some cases the artifact clusters were known, so we can compare to the known clusters. Or perhaps, someone was paid to cluster some images into segments by hand. It's still not quite as easy to evaluate the clustering as a classification would be, since any permutation of the labels leads to an equally good clustering.

The only thing that defines the clustering, is whether two data points belong to the same cluster or to different clusters. Measurements like the Rand Index and Adjusted Rand Index, help us measure whether the clustering found by our algorithm, really captures the information in the ground truth clustering. They do so, by counting the pairs of data points that belong to the same or different clusters according to algorithm.

And the same or different clusters according to the ground truth. Then they normalize appropriately. Measures like the Rand Index are called External Evaluations, because they require outside information about a ground truth clustering. The problem of evaluation is typically even trickier in clustering, because there's rarely any ground truth information that we can use for testing.

If we already knew the clustering, we'd be done. So how do we evaluate whether a clustering is good, or whether one clustering is better than another? You might think, well, we can check with the value of the K-Means objective function is, across different clusterings. This will let us compare two clusterings, but it doesn't tell us whether a particular clustering is good or not on it's own.

It also might not capture exactly the patterns we're really trying find in the data. As we'll see later in this video, as well as in future videos. So how do we evaluate whether a clustering is good? The short answer is, no one agrees. But the longer answer is, that researchers have developed a number of useful heuristics.

The first, and perhaps most useful observation, is that we're often trying to find some particular, meaningful latent pattern in our data via clustering. Since we have a conception of what that pattern is already, we might be able to check if we found it, by visualizing the results of the clustering.

For instance, when you saw the archaeology data, it was obvious what the clusters should be. Similarly, consider again, the image segmentation example. We would like our clustering to distinguish the three different types of objects in the image. And we could easily check this by visualizing the clustering of pixels.

Or, consider the topic modeling example, where we clustered words together. We can list out the clusters of words, and ask ourselves, do these words go together? Do they form a cohesive topic? Or, are they words that don't really have a meaningful association? Another option is to use special tools for visualization, like the GGobi tool.

We might have a data set, where D, the dimension is much higher than two. For instance, we might have collected data on a large number of a different health metrics. Like age, daily calories consumed, daily water consumed, weekly alcohol consumption, weekly miles driven, weekly exercise in minutes and so on.

But this data isn't an image or a text data set. So it might be hard to plot meaningful pictures of the data. Tools like GGobi, help us find meaningful visualizations of high dimensional data for evaluating the clustering. While these methods are often more qualitative than quantitative on their own.

They can be made more quantitative by incorporating some form of Crowdsourcing. For instance, Amazon Mechanical Turk workers could be asked to weigh in on the quality of the clustering. But even with the help with Amazon Mechanical Turk or other crowdsourcing platforms, human evaluation can be expensive in terms of both time and money.

So it's worth considering other automated forms of evaluation. By contrast to External Evaluations, there are a number of measures for Internal Evaluation that depend only on the data at hand. The basic idea of these measures, is typically to make sure that data within a cluster are relatively close to each other.

And data in different clusters are relatively far from each other. An example of this, is the Silhouette Coefficient. Another example is, to split the data set into two data sets. Applying clustering to each and comparing the clusterings found across the two sub data sets. There are also a number of measures for evaluation that are specific to certain domain or application.

In this video, we've discussed some of the ways to evaluate the output of the K-Means Algorithm and Clustering Algorithms more generally. So, what if I evaluate my output and I'm not totally satisfied? What are some potential problems with the K-Means Algorithm and how can it be improved? We'll answer these questions in the next few videos.

1.8 Beyond K-means: What Really Makes a Cluster

Welcome back. In the last few videos, we introduced the K-Means clustering problem, and a particular algorithm for coming up with a solution to it. This algorithm is called the K-Means algorithm or Lloyd's algorithm. Then we talked about how we might start inspecting the output from the K-Means algorithm.

So what happens if that output isn't what we expected or wanted? What do we do now? In this video, and the next few videos, we'll talk about troubleshooting K-Means. How can we make K-Means clustering better? In answering this question, we'll often find ourselves leaving K-Means behind for different clustering algorithms, and even different clustering problems.

The first thing to do though, is to make sure we're getting the best possible output from the K-Means algorithm. Now we know that the K-Means algorithm will stop eventually. To see that this is true, note that there are only finitely many possible clusterings. And the K-Means algorithm will move to a new clustering only if the new clustering decreases the K-Means objective.

But just because the algorithm will top eventually, doesn't mean that the algorithm will stop quickly. That being said, in practice the K-Means algorithm tends to be quite fast, but you might find that on a particular data set, the algorithm is taking a while to run. What do you do?

There are various speed ups that have been proposed. These are often based on the following idea. In the K-Means clustering algorithm, as we wrote it, we needed to calculate the distance from every data point to every cluster center. But you don't actually need to check every possible cluster center to decide which is the closest for a given data point.

This observation relies on a fact known as the triangle inequality that lets us ignore cluster centers that are relatively far away from a given data point in our calculations. Another related issue is that when the K-Means algorithm stops, it doesn't necessarily stop at a minimum value of the K-Means objective.

Remember, this objective is the global dissimilarity. When the K-Means algorithm stops at a value that isn't the value of the K-Means objective, we call the objective value it does stop at a local optimum. We wish we had access to the true minimum, the global optimum. But that's a very difficult problem that we can't solve in general.

One option to try to get closer to the global optimum is to run the K-Means algorithm multiple times through multiple random initialization. Then finally, we take the configuration of cluster centers and cluster assignments with the lowest objective function value, that is, the least global dissimilarity. While this does require more total computation, these multiple runs can be performed completely in parallel.

Moreover, another option to get better output from our algorithm, is to use smarter initializations. Like in K-Means++. This also has the potential to reduce total running time. In fact, K-Means++, unlike Vanilla K-Means, comes with theoretical bounds on the total running time. Okay, but imagine we somehow magically had access to the global optimum.

We could know the actual set of cluster of centers and assignments of data points to clusters that minimize the K-Means objective function. And we could still be dissatisfied with our results. In the upcoming videos, we'll talk about some examples of how this could happen, and what we can do about it.

In this case, the issue isn't our K-Means algorithm, the issue is the K-Means clustering problem itself. In this video, we focused on how to improve the K-Means algorithm to get better results when it comes to minimizing the K-Means objective. But we also hinted that there might be deeper issues with the K-Means clustering problem in certain application domains.

Recall that clustering is grouping data according to similarity. In the next few videos, we'll dissect how each of those terms might have a different meaning than the one we assume in the K-Means clustering problem. Next, we'll start by seeing a number of different ways we might define grouping.

1.9 Beyond K-means: Other Notions of Distance

Welcome back. In the last video, we started to hint that there might be various reasons one might want to go beyond the K-Means Clustering problem. In this video, we'll see some concrete examples of when that might happen. As we said last time, clustering is grouping data according to similarity.

In this video, we'll examine different notions of grouping than we've seen so far in the K-Means Clustering problem. Probably the most immediate and obvious issue with K-means clustering, in a variety of applications, is that it requires the user to specify the number of clusters, K. Sometimes this is completely fine.

For instance, K-means can be used for image compression. In this application, each data point might be the red, green, and blue intensity values for each pixel and the K means algorithm finds K colors that can be used to approximate the true, and typically much wider, range of colors in the image.

Here, K might be determined by how much we want to compress the image. In this illustration, we see how increasing K increases the quality of the compressed image. In many other cases though, K is unknown in advance. Suppose we get some new data and we need to determine both K, as well as the latent clusters.

One option is to use popular heuristics. One heuristic arises from plotting the global dissimilarity across a wide range of values of K. We can't choose K immediately from such a figure, because the K-means objective function will always decrease as we increase K. If we just choose K as the value that minimized the K-means objective function, we'd have to choose K = N, the number of data points.

But that's a useless clustering. An alternative heuristic is to look for an elbow in a plot of global dissimilarity as a function of K. That is, a point in the figure where the gain from new clusters suddenly slows down. We can set K to be the number of clusters at this elbow.

Other, more theoretically motivated options for choosing K exist, as well. These options include the gap statistic as well as methods that change the objective of K-means. The idea of these latter methods is to explicitly account for the fact that we pay a price for more complex models. In particular, we can start with the original K-means objective function and add a penalty term that grows with the number of parameters.

Some appropriate penalties are given by AIC, BIC, or other so-called information criteria. Once these penalties are added, we can plot the new objective function as a function of K, and now there will typically be a non trivial minimum. We can choose K to be the number of clusters at this minimum.

There are broader issues with choosing K though. Sometimes there isn't a clear right value of K. Suppose we have a bunch of data on various organisms we've studied in some environment. We might find that if we favor smaller clusters, we cluster the organisms roughly into species. Which is great.

We've picked up an interesting and meaningful latent grouping of the organisms. But if we favor larger clusters, we might end up clustering the organisms at the genus, family, or order level, which are all useful and meaningful classifications used in biology. Clustering at any of these levels would lend insight into the problem, but each level would result in a different number of clusters.

In this case, it might be useful to try a hierarchical clustering approach, such as agglomerative clustering. These approaches explicitly model that some clusters might be composed of further sub-clusters. Finally, all of the clustering we've talked about so far, puts each data point into one and only one cluster.

This is called hard clustering. On the other hand, we might sometimes have clusters that aren't perfectly separated. Some data points might be on the border between two or more clusters. For instance, recall our archaeological dig example. You might have an artifact that happens to be close to the cluster centers corresponding to two different families.

When we go back to the lab and analyze this artifact, it would make sense to keep in mind that we're not sure which family it belonged to. More broadly, we'd often like to express our uncertainty about which cluster a given data point belongs to. And alternative to hard clustering that allows us to do this is soft clustering.

In soft clustering, we might allow each data point to have a different degree of membership in each cluster. For instance, a data point might have a probability distribution over its belonging in different clusters. For many data points, this probability distribution might express that we're very sure that the data point belongs in one particular cluster.

But in other cases, nearer the border between clusters, the probability distribution for a data point might favor one cluster, but also make it clear that the data point could reasonably belong in another cluster. In the archeology example, we might have a best guess as to which family generated some piece of pottery.

But we want to encode that we're not certain and another family could reasonably have made and used the pottery. In the soft clustering case, we might use alternatives like fuzzy clustering and mixture models such as Gaussian mixture models instead of K-means clustering. In this video, we saw a number of examples with a different notion of clustering that the notion encoded in K-means clustering.

We saw that we might not always know the number of clusters before we apply clustering. Moreover, we saw that clusters can sometimes be at multiple scales and even nested within each other. Finally, we saw that clusters don't have to be an all or nothing proposition. Often, it's useful to express that we're uncertain about the cluster assignments of some data points.

Here, we've explored the notion of what makes a cluster and it has motivated us to look at clustering problems and models beyond K-means clustering. In the next video, we will similarly dissect the notion of similarity.

1.10 Beyond K-means: Grouping Data by Similarity

Welcome back. So far we've seen the k-means clustering is a powerful framework for clustering. But in the previous videos, we've started to explore how k-means clustering isn't always the best clustering framework for a particular problem. So sometimes it behooves us to look at clustering frameworks beyond that of k-means.

We recall that clustering is grouping data by similarity. In the last video, we looked at how we might be interested in different notions of grouping or clustering beyond that provided by k-means. In this video, we're going to look at how we might sometimes want to define similarity differently than the definition used by k-means cluster.

It's worth recalling that k-means clustering assumes that we have K spherical, equally-sized clusters. This follows from our use of the Euclidean Distance as the dissimilarity measure for K means. Using the squared Euclidean Distance in our objective function is equivalent to using Euclidean Distance and doesn't change this fact.

So what could go wrong with the k-means definition of similarity. First, let's consider the case where our clusters aren't equally sized. For instance, consider an example where we have small circles filled with data points adjacent to a very large circle filled with data points. We might naturally think that there are three clusters here, corresponding to the three circles.

In the k-means objective though, the clustering where all of the data points in the larger circle are assigned to the same cluster, has relatively higher or worse objective value. And the clustering where the data points in the large circle closet to the smallest circles are assigned to the outer clusters, has a relatively lower or better objective value.

This is because the data points at the edges of the large circle are relatively far from the clustered center there, but relatively close to the cluster centers in the small circles. What can we do if we really want to separate out the three circles as clusters? One option is to use a model that specifically encodes clusters of different sizes.

For instance, in the last video we mentioned Gaussian Mixture Models. When the mixture components are allowed to have different co-variances, these models can capture the kind of clusters we're looking for here. One useful algorithm for this type of model is called the expectation maximization or Algorithm. Another issue that might arise is that we might encounter outliers in our data.

Outliers are data points that are relatively distant for most of the data points in the data set. Because of its use of Euclidian distance K-means clustering is typically very sensitive to outliers. Consider this simple one dimensional dataset. If we run the K-means algorithm with K=2, we discover that there are two clusters as we expect.

But if we add a single point very far away from our data, we're now stuck with one cluster over all the main dataset and the second cluster contains only the outlier point. This clustering doesn't capture the two clusters we think really exist in this data. What can we do?

One option is to use an alternative measure of distance that is less sensitive to outliers. To calculate the square Euclidian distance, we added up the square difference between every pair of features shared by two data points. This is sometimes called the L2 distance. An alternative distance is the L1 distance, where we say that the distance is the sum of the absolute values of the differences between every pair of features shared by two data points.

Pretty much everything we already did can be adapted by this case. In particular, we can derive a iterative algorithm like K-means for this problem. The algorithm is called K-Medoids since the optimal clustered centers are now medoids instead of means. The medoid in one dimension is the familiar medium.

Finally, we might be interested in clusters of different shapes than spherical clusters. Consider this example data set. Here, there seems to be two identifiable groups, or clusters of points, but one group is situated inside the other. If we applied k-means with two centers, we'd just split the data into two sides of some essentially straight line.

To capture the two clusters we think really exist here, we'd have to try something different. For instance, we might define a radial notion of similarity. Consider splitting the data into polar coordinates, or use kernel methods. Another alternative is to use the agglomerative clustering method that we mentioned in the previous video.

Another potential issue with applying K-means is that it requires continuous numerical feature. If we're dealing with text or binary yes no features, or other features that aren't continuous and numerical, you might consider alternative notions of similarity. Another option though is to transform our data into a form more amenable to K-means cluster.

We'll talk more about this in the next video. In this video, we explored using different notions of similarity and clustering and the square Euclidian distance required by k-means clustering. In the previous video, we looked at different notions of clustering and clusters than those required by k-means. In the next video, we'll look at what we can do when our data isn't already in the form assumed by k-means.

1.11 Beyond K-means: Data and Pre-Processing

Welcome back. In the past few videos, we've been talking about troubleshooting the k-means algorithm. In particular, we've been working our way through the definition of clustering as grouping data according to similarity. We talked about what happens if we have a different notion of grouping than k-means. And we talked about what happens if we need a different notion of similarity than k-means.

In this video, we'll take a closer look at our data. When we first step through the k-means algorithm we imagined, as an example data set, that we were looking at the locations of archeology artifacts. In particular, for each data point we recorded a distance east of some marker, and a distance north of the same marker.

But there are a lot of other types of data out there that I might collect. How do I know when it's okay to run k-means? And is there anything I can do to make it possible to run k-means? Before running k-means on your data, there are a bunch of questions you want to ask yourself.

We'll step through the big ones here. First question, is your data featurized? As an example, suppose you've made a fitness app and you've collected a bunch of data about your users. This person is 45 years old, 5'7" tall, has a terminal bachelor's degree, occasionally post on public forums in your app and so on.

Remember that k-means needs a vector of data. So the first thing we need to do is turn all of this information into an orderly vector. Each entry in the vector will have a particular meaning. The first entry could be a person's age. The second entry could be a person's height.

The third could be a person's education. And the fourth could be how many ounces of smoothie this person drank in the past week. And maybe there're a bunch more entries. Each of these entries is called a feature. It's a feature, or a trait, of the data with a very specific meaning.

We can imagine a big table or a matrix that collects these features across all of the users of the app. Once we've organized our data like this it is featurized. Next, remember that k-means needs its features to be valued as continuous numbers. So the second question we have to ask ourselves is, is each feature a continuous number?

In our example, age as in years, so we're all set there, we can convert height into inches. So a 5 foot 7 person would be 67 inches tall. We could express education as years of education and write 16 for a bachelor's degree. And 8.2 oz of smoothie is also fine.

The third question gets at a subtle point about k-means. Are these numbers commensurate? To see the issue here, recall that k-means assumes we have spherical clusters. So the clusters have basically the same width in all directions. Imagine we have the following data set, but notice the axes. One axis ranges from 0 to 10, and the other ranges from 0 to 1.

So if we plot the axes on the same scale, this is what the data looks like to the k-means algorithm. So we might want to put all the data on the same scale. And typically a standardization or normalization procedure is used. For instance, all data in each feature can be separately standardized to align between -1 and 1.

A common alternative is to normalize the data and each feature to have a standard deviation of 1. Another subtle question is the fourth question, are there too many features? If I have a fitness app, I might want to find different groups of individuals who have similar health and fitness needs.

But suppose I add a bunch of useless features, like for every word in the English dictionary I add a feature for how many times this particular person uses that word in the forums. This sounds silly, but a lot of times we have features in our data that aren't entirely useful to the purpose at hand.

When we know which features aren't useful, because of some domain knowledge, we want to get rid of them beforehand. But sometimes we don't know which are the useful and useless features. In this case, there are some pretty magical algorithms that let us reduce the dimension of our feature space.

That is, they let us reduce the number of features to just get the most important features out. Probably the most widely used algorithm of this sort is principal components analysis, or PCA. As an example, imagine I have a spring and a ball is moving back and forth on that spring.

And I record this motion with a bunch of different cameras, maybe five cameras. Well, the motion of the spring is really just one dimensional. So I effectively have four extra features. PCA will help me pick out the one feature that really matters. Even though it doesn't even necessarily correspond to just one camera.

So it's often a good idea to run PCA on your data before running k-means. In this case, we refer to PCA as a preprocessing step for k-means. Okay, a fifth and final question to ask is, are there any domain specific reasons to change the features? As I mentioned earlier, it's best to get rid of any features you know in advance won't be helpful for the final problem.

Typically you or a colleague might have this knowledge due to being an expert about the particular data you're using. Note that any domain specific feature changes should be done before running PCA or k-means, or anything else for that matter. It's also best to change any features that might not satisfy the assumptions of k-means.

For instance, as we talked about in both this video and the last video, there might be transformations of your features to something that better fits the assumptions of k-means. It's nice to automate everything about using machine learning and statistics. But if there's knowledge you can use specific to your problem area, it can sometimes make a big difference if you use it.

In this video, we talked about preparing, or preprocessing your data set so that you can run the k-means algorithm with the best possible results. In the past few videos, we've started to dig deep into the grouping, similarity, and data assumptions that go into k-means clustering. In the next two videos, we'll explore further the assumption of a fixed number of clusters, k clusters, and clustering itself.

1.12 Beyond K-means: Big Data and Nonparametric Bayes

Welcome back, in the past few videos, we've been examining some of the assumptions that go into k-means clustering. And we've talked about what you can do if you want to cluster data, but you application or your data don't quite fit those assumptions. In this video and the next though, we're going to take a step back.

After all, k-means clustering is just one type of particular fixed-k clustering. That is, clustering where we assume the number of clusters is finite. Even when we don't know the number of clusters in advance, we often still assume that the number of clusters in our data is some fixed, finite number.

In this video, we'll talk about why that assumption might not always be right. Often, big data doesn't just mean that we have a single, very large data set. We might have streaming data where new data is being added to our data set all the time. For instance, English language Wikipedia alone averages around 800 new articles per day right now.

If we build a new fitness app we hope that it will not only get new users all the time but that those users will interact with it everyday. So what changes when we have streaming data? To illustrate, I like to think about my own experiences with Wikipedia. I recently read this really neat book, A Long Trek Home about a couple who travelled 4,000 miles from Seattle to the Aleutian islands in Alaska by hiking and rafting.

So this makes me go on Wikipedia and look up the Aleutian islands. And that leads me to an article on the Pacific Ring of Fire. And soon I'm reading about volcanoes, earthquakes and plate tectonics. And the thing about Wikipedia is, no matter how many articles I've read in the past, there are always new topics to read about and learn about.

I like to think of this as some sort of Wikipedia phenomenon. Suppose I have a hundred articles from Wikipedia. I could cluster those articles and find some topics to group articles together. But if I have 1000 Wikipedia articles, I expect to find new topics I didn't see in the first 100 articles.

And as I read more and more Wikipedia, getting to millions and billions of articles, I expect to keep finding new topics. In this case, it might be okay to run clustering with some fixed number of clusters for 100 articles, or any fixed number of articles. But as the size of my data set grows, I always expect to see more groups, more clusters.

So I want the number of clusters to grow with the size of my data. Or another way to think of it is, I want the complexity of my model to be able to grow with the size of my data. We see this in a lot of other types of data as well.

Humans have been studying and discovering new species for a very long time now. And yet, at anytime, you can do a search on Google News and be sure to find all kinds of new species that were discovered in the past week. Humanity has discovered a ton of different species, but we're still discovering new ones all the time.

And we're not done yet. Thinking back to our imaginary fitness app, we could imagine asking people about their exercise routines. And no matter how many people we ask, we might expect that we will keep discovering new and unusual exercises that we haven't seen among the previous users we'd asked.

In genetics, we might expect to find new and unknown ancestral populations as we study more individuals' DNA. Or suppose we look at newborns in a hospital. As we study more and more newborns, we might expect to find more new and unique health issues among those newborns. And we want to be prepared for these new health issues so that we can best treat these newborns and not miss important diagnoses.

We might consider a social network. As more and more people join the social network, we'd expect to see new friend groups and interests represented in the network. Or we can imagine image database like the images on Instagram. As we examine more and more images, we expect to find more unique objects in the images.

Objects we haven't seen before in previous images. In all of these cases, we wouldn't want a fixed number of clusters, k for clustering. We want k to be able to grow as the size of the data grows. One solution to this problem is provided by nonparametric Bayesian methods.

Nonparametric makes it sound like there are no parameters. But in the context of the term, nonparametric Bayes, nonparametric actually means many parameters, or, infinitely many parameters. These methods let the number of instantiated parameters grow with the size of the data. While these methods are more complex than k-means clustering, some recent work on MAD-Bayes shows how to turn nonparametric Bayesian methods into k-means-like problems and algorithms, for clustering in particular, and unsupervised learning in general.

In this video, we've talked about why you might not always want a fixed number of clusters k. Even when k isn't known in advance, the assumption that k is the same throughout some streaming data set, might be problematic. In the next video, we'll talk about cases where you want to find hidden patterns in your data, but clustering might not be quite the right type of pattern.

1.13 Beyond Clustering

Welcome back. In the previous video, we started talking about cases where traditional clustering just doesn't cut it anymore. Last time, we looked specifically at the case where we can't assume there's a fixed number of clusters in our data set. We might want the complexity of our model to grow with the size of the data.

In this video, we'll talk about cases where clustering doesn't quite represent the hidden patterns we want to find in our data. Let's think back to the image analysis example from last time. For the moment, suppose that I have a website where people post pictures of their pets. I can imagine that I might like to cluster these images.

And I'd be very happy if my clustering algorithm put all of the images of cats in one cluster, all of the images of dogs in another cluster, and all of the images of lizards in yet another cluster. One way I could represent this clustering is in a table or a matrix.

Each row of the table or matrix will correspond to one of my images. I'll put a one in the column that corresponds to the cluster, this image belongs to. So here, pictures one, two and seven belong to the same cluster. Pictures three and five belong to the same cluster, and picture four is in it's own cluster.

But it seems something's missing. What if someone post a picture of both a cat and a dog, or a cat, and a dog, and a lizard? Or what if they post a picture that doesn't happen to have any animals in it? The problem with clustering is that each data point has to belong to one and only one group or cluster.

We might express uncertainty about which group that is. But ultimately, we believe that the ground truth is that there's exactly one true group that the data point belongs to. So now, think back to our website and people's pet pictures. Here, we might want our data points, the pictures, to belong to multiple groups, simultaneously.

Or no groups, or just one group. Any of these options should be allowed. In this case, we call the groups features instead of clusters, and the underlying structure is a feature allocation instead of a clustering. Note that this is a different use of the word feature than we saw in previous videos.

A similar idea is to say that the data points exhibit mixed membership. They can belong to multiple groups at the same time, unlike clustering. Sometimes, clustering is called a mixture model since each data point comes from one of a bunch of different groups. The bunch of different groups forms the mixture.

Then, the case where each data point can belong to multiple groups at the same time is called an admixture model. The word comes from the genetics literature. But essentially, all of these terms feature allocation, mixed membership and admixture capture the idea that data points can belong to multiple groups simultaneously.

We've seen this sort of motif crop up in lots of different data. Suppose we're analyzing a corpus of documents and we want to find the topics or themes that occur. We might look at all of the documents in past issues of the New York Times. Some natural topics that we might find could include sports, arts and economics.

But then suppose we read a review of the movie, Moneyball, based on the book by Michael Lewis. Well, it's about the arts because it's a movie review. The review is also about sports because the movie is about baseball players. But the review is also about economics because the movie is about using analytics and statistics to trade players and choose the best players.

So if we think of a document as a data point, we want our data points to be able to belong to multiple groups. Similarly, if we study genomics, we often see that an individuals DNA may be comprised of parts from a number of different ancestral groups. If we study politics, we may see that individuals votes actually represent a number of different political ideologies.

If we study social networks, we may see that an individuals interactions on a social network represent various different personal identities, such as a work identity, a family identity, and a social identity separate from the first two identities. There are a number of different models and algorithms that let us go beyond clustering and capture this kind of mixed membership structure in data.

Perhaps the most popular of these algorithms is Latent Dirichlet Allocation or LDA. LDA was originally designed for text data and it's popularity may be due in part to the rise of massive amounts of text data available online. But it can be applied much more widely to other types of categorical data, including genetics data.

Other K-Means-like algorithms for this feature allocation problem have been derived using Mad-Bayes. For instance, the DP-Means and BP-Means algorithms have been used to study tumor heterogeneity. In many tumors, multiple different types of cancer are present. It's important to identify all of the different types of cancer so as to design effective treatments.

In this sub-module, we've studied clustering in general and the very popular K-Means algorithm for clustering in particular. In the past couple of videos, we've gotten a taste of when and how you might start to go beyond the clustering paradigm. In the remaining videos of the sub-module, we'll get some hands-on experience with clustering and mixed membership.

Not only will you know all about clustering and K-Means and its extensions, but you'll see them used on practical problems. But it's worth emphasizing, there are many other forms of unsupervised learning out there beyond the ones we've seen so far. Clustering and mixed membership, and feature allocations are just the beginning.

Case Study 1.1: Genetic Codes

Welcome back. In the previous videos, we've introduced unsupervised learning. We saw that clustering is a particular form of unsupervised learning and that k-means clustering is a particular form of clustering. In this video we'll, explore k-means clustering in an extended application due to Alexander Gorban and Andrei Zinovyev, Understanding the Genetic Code.

Recall that DNA is, essentially, a sequence of letters. A's, C's, G's, and T's. You typically learn in a biology class that these letters form a sort of code, a set of instructions for how to grow and maintain a living organism. But how can we interpret the sequence of letters as a set of instructions?

Well, any English sentence is a sequence of letters. And the units of meaning within that sentence are words. So we might hypothesize that a DNA sequence is similarly composed of words. A particularly simple hypothesis, is to assume that the words are all of the same length. We might first guess that the words in DNA are either one, two, three, our four letters long.

So the question we want to answer is, is it true that DNA breaks down into meaningful words of the same length? And if so how long are the words? We're going to see how we can use k-means to help answer these questions. First, we gather up some real DNA.

A fragment of the full DNA sequence of particular bacteria. The total fragment is made up of around 300,000 letters but you can try any other DNA sequence instead. Next, we break this full sequence into strings of 300 letters each. And we make sure these strings don't overlap at all.

We'll think of these strings like a data point. So in this data set, we end up with 1,017 data points. Let M be the proposed word length. So with M=2, we expect each word to be two letters long. For each value of M, we're actually going to make a different data set.

Recall that in order to run k-means, we need to featurize our data. That is, we need to come up with vector of numerical features for each data point. Here's something we can do. For any value of M, we can divide the DNA string up into substrings of that length.

For each possible word like, AC, we'll count how many times that word occurs. That is, how many substrings, are the same as that word? Consider M=2 for a moment. How many possible words are there? Well, there are four possible letters in the first slot, and there are four possible letters in the second slot.

So, there are 4 squared, or 16 possible words. So, for the M=2 data set, each data point will have 16 features associated with it. For instance, the AC feature will tell us how many times the two letter word AC occurs in the string. Since these counts only take integer values, they're not exactly continuous values, but they're close enough.

Okay, so our data set for words of length 1 has 4 features, our data set for M=2 has 4 squared features, our data set for M=3 has 4 cubed features, and our data set for M=4 has 4 to the 4th features. That's a lot of features. Even for M=1, four features is hard to visualize.

As humans, we basically only ever look at plots in two dimensions. Even three dimensions is pushing it. So, four features is already a lot. And it just get's worse as the proposed word lengths get longer. Since we can't really visualize these data sets as they are, let's run PCA.

Remember that PCA stands for principal component analysis. We'll pick out the top two features from PCA. Also remember these don't have to correspond to any two features in our data set. They can be wholly new composite features. Julia Lye, an awesome engineering student, did the coding and plotting for this case study, as well as the next.

Here Julia plotted these two PCA features for each data set. We immediately see something really interesting here. There's not much going on for M=1 or M=2. The data just form a big blob. And that's also pretty true for M=4. But there's a lot of clear structure and symmetry for M=3.

So from here on out, let's concentrate on M=3. And now let's run k-means. Remember, we still need to standardize or normalize our data somehow. So let's first normalize each feature by subtracting the empirical feature mean across the data set, and dividing by the empirical feature standard deviation. Now, we'll take that normalized data and run k-means.

How should we choose k? Well, from visual examination of the PCA plot from earlier, it looks like there are six or seven clusters. Let's try each value of k and see what happens. First we try k-means with K=6 and we visualize the end result using the first two components of PCA.

Remember, we ran k-means on the full 4 to the 3rd, or 64 features here. We didn't just run k-means on the two dimensions that we're visualizing. So it's a meaningful and great sign that the k-means results align so well with the structure we're seeing in this visualization. Now let's try K=7.

In this case, we pick up even more structure. We're able is almost perfectly separate out the middle part from the six lobes around the outside in the visualization. So what's going on here, what does this result mean? Well, let's talk a little bit more about DNA. Suppose the words are really length three, as we've been assuming and running k-means on the M=3 data.

Then what would we expect to see? Well, we don't know exactly where these words start, so there are three possible starting points we might pick up. When we divided up DNA into words, we might have started on the first letter of the word, the second letter of the word, or the third letter of the word.

And actually, we don't really know that all the genes read forward for the direction we chose for our strings. Some might read backward too. So we might start on the first letter backward, the second letter backward, or the third letter backward. And finally, there might be some noncoding regions of DNA.

DNA that doesn't actually have any instructions and acts as filler. We've seen from the other M plots, like M=1 and M=2, that filler kind of looks like a center blob. And that's exactly what we're seeing here. We see six lobes, one for each of the possible directions we might read meaningful words in DNA.

Three directions forward and three directions backward. And the seventh blob in the middle represents words in noncoding DNA. In fact, now that we've run k-means, we can go back into each of our data points, each of our DNA string segments, and say whether it's noncoding or coding. And which of the sixth shifts it shares with other strings, based on the results of k-means.

So we've seen strong evidence that DNA is composed of words of three letters each, as well as noncoding bits. In fact, these three letter words are now known as codons. And scientists know that they basically encode amino acids. We were able to show this with just a data set of a DNA sequence fragment.

And we're also able to identify which of our strings are likely coding regions and noncoding regions. And which have similar offsets. In this case study, we've seen how quickly and easily we can use k-means clustering, and PCA, another powerful unsupervised learning method, to get results that lend deep insights into a scientific problem.

In the next case study, we'll look at another data type that is ubiquitous these days, human-generated text.

Case Study 1.2: Finding Themes in the Project Description

Welcome back. In the previous video, we looked at using k-means clustering in practice to try to understand DNA as a genetic code. In this video, we'll look beyond k-means and clustering to see how some of the extensions we discussed earlier can be useful in real life data analysis.

In particular, here we'll analyze human generated text data. It's common nowadays to have access to large text corpuses. The text of Wikipedia is publicly available. We can look at the text of news releases from MIT. But text data can arise in many other forms. A news website might have the text of all of its articles, and the company running the website might be interested to find natural ways to present this text to its audience.

It might be helpful to present topics in the text for easy navigation to relevant news articles. Or a search engine might be interested in analyzing the text of websites it crawls for convenient presentation of its search results. For instance, Michael Jordan is the name of both a famous basketball player, and a famous statistics researcher.

Suppose someone enters Michael Jordan as an input into our internet search website. If we presented search results by topic instead of solely by popularity we could quickly reach both the audience searching for sports information as well as the audience searching for scientific research. So there are a lot of reasons we might be interested to find the topics or themes in a collection of text.

But as we mentioned in a previous video, clustering by itself can be a little problematic here because a single document might exhibit multiple topics. What if Michael Jordan, the statistician, writes a statistical analysis of the other Michael Jordan's basketball career? Or what if I look at the Wikipedia page for the sports figure Michael Jordan?

Actually, Wikipedia covers not just his basketball career, but his brief time playing baseball, as well as his experience as a businessman, and his acting. Remember Space Jam? So it makes a lot of sense to use a model that let's each data point, each article exhibit multiple topics. Recall that one term for this is a mixed membership model.

A particularly popular model in this vein is known as Latent Dirichlet Allocation, or LDA. To see what it looks like to use this model in practice, let's use LDA to analyze what MIT faculty are working on in their research. The coded visualizations for this analysis were all provided by

Julia Lott.

The electrical engineering and computer science program at MIT is really big. It's the biggest department on campus, with over 130 faculty and there are a number of major research labs that those faculty belong to. Four of the main labs in EECS are the Computer Science and Artificial Intelligence Laboratory, or CSAIL, the Laboratory for Information and Decision Systems, or LIDS, the Microsystems Technology Laboratories, or MTL, and the Research Laboratory of Electronics, or RLE.

If you've never heard of this labs before it's not necessarily clear what they all do. And even if you're a member of this department it's hard to keep track of the research of 130 other faculty members. But you might want a high level summary of what everyone else is working on.

For instance if some area of research looks interesting you can go talk to the faculty working in that area. This line of thought suggests we should try to find latent groups of research topics among the faculty. But a faculty member can work on multiple research topics at a time, so we're going to try to learn a mixed membership model like LDA.

To create our dataset we assemble abstracts from each professor's published papers for a total of over 900 abstracts. Each abstract is a text document that briefly describes a particular piece of research the professor conducted. A common pre-processing step for LDA is to remove the most common words, words like of, the and a that don't tell as much about a document.

And the least common words, words so rare that they don't tell as much across documents. Just like we need to specify the number of clusters in advance in k-means we need to specify the number of topics in advance for LDA. And just like we called the number of clusters K, for clustering, here we can call the number of topics K.

In an earlier video, we talked about heuristics to choose the number of clusters in a clustering problem. Similar heuristics can be used to choose the number of topics in topic modeling. Heuristics for this problem suggest that smaller k yield more meaningful results, and henceforth we choose k equals 5 topics.

At this point we can run a stochastic variational inference algorithm for LDA and look at the results. Just as in clustering we discover clusters, in topic modeling we discover topics. So the first thing to do is to look at the topics. We can visualize topics by looking at the most common words in each topic.

In this case, one of the topics has words like quantum, state, system, channel, and information. This topic seems to be capturing a line of research on quantum computation communication and information theory. Another topic has words like algorithm, model, problem, n, and time. This is a topic on traditional computer science themes, like algorithms and complexity theory.

N is a symbol typically used to express the number of data points in a data problem. Yet another topic has words like temperature, graphene, phase and gate. These are words related to material science. So one of the first things we notice here, is that research in the department is pretty diverse.

Which makes sense, since the department contains both electrical engineering and computer science. We can also look at the topics of research in each lab. To generate this figure, we consider all of the research abstracts within each lab and show what proportion of the research text falls into each topic category.

First, we notice that CSAIL and LIDS are both dominated by the algorithms topic. This makes sense, CSAIL essentially contains all of the computer science professors within the EECS department. And the faculty and LIDS work on similar problems with perhaps a bit more focus on information theory. We see this reflected in the higher proportion of the topic on quantum and information related themes.

RLE and MTL both contain words related to circuits, again this make sense since these labs form the more traditional lab-based side of electrical engineering. Finally, MTL is distinguished by a significant proportion of research text related to material science. So the nice thing about this analysis is that we learned all of these topics and the breakdown of labs by topic automatically.

Few people, including professors at MIT, would be able to describe exactly what all these different MIT professors work on. But we were able to automatically pick up what words go together and describe coherent themes of this research. And we were able to distinguish different parts of the department by the different types of research that goes on there.

You could imagine doing a similar analysis for a company or for the user base of an app. What are the different themes users talk about in their posts on a forum? And how do the different forums differ in what they focus on? This type of analysis can be very useful for understanding a user base or for generating a quick summary of a lot of text documents that might otherwise be difficult to navigate.

In this video we've seen how extensions of clustering can be useful in practice. In particular we investigated the data analysis problem with human-generated text using latent dirichlet allocation or LDA. We've come a long way in this module and sub-module. We started by introducing unsupervised learning, and then we focused in depth on clustering and k-means clustering in particular.

We saw all the ins and outs of using k-means clustering in practice. And let me be clear, this sort of detail and care is necessary for any machine learning or statistics algorithm. No algorithm is a black box. You want to understand what assumptions are going in to your algorithm, and there are always assumptions.

Once you understand these assumptions you can decide if your algorithm is the right approach to a problem or if you need an extension or something totally new.

1.2 Dimensionality Reduction and Spectral Techniques

1.14 Networks and Complex Data

In the past few sections we have already seen many interesting and very useful methods to find grouping structure in data. For example, the k-means algorithm. In the next few sessions we'll see other very interesting tools that tell us a whole lot about our data. In particular some of these work even if we can't apply k-mean.

So let's quickly recap a few things. For clustering and many other tasks, our data comes in form of data points. Each data point is a feature vector, you can think of it as a string of numbers and each number describe a feature. For example, in the introductory email example, each email is a data point, it is described by the words contains.

So we have a huge vector and each entry in this vector corresponds to a word. The number for each word means how often this word occurs in the email. With such representation we could use k-means, but then we are not always so lucky. Let's think about some examples.

The feature vectors can contain too much useless information or noise. We may not have such vectors at all. In such cases we may be able to construct new features from whatever we have. Sounds weird? Well, soon we'll see how to do that. First let's look at some examples.

Data can contain a lot of measurements. That is, each data point is a huge feature vector with many entries. Think of a person being a data point and the descriptions, the variations in the person's genome. That can be a lot. Or think of a collection of images that are face portraits.

Each image is a data point and is described by a few hundred thousand pixels. But then some pixels will be more meaningful than others. If you look at the upper right corner of face portrait images, that is most likely going to be not so relevant. It is constant or varies randomly across images since it's background.

So there's not a whole lot of information. The important information is how the images deviate in a major way from some average image. For example, there will be variations in hair, glasses, beard and so on. These major axes of variations are what captures the data in a more meaningful way.

And maybe there are actually much fewer axes than pixels. Why? Some groups of pixels tend to vary together. Because they belong to some sub-region, like eyebrows or chin. These can help compress the data. Similarly, user ratings can be much more easily understood with patterns. My ratings for a movie can be understood more easily if you know that I like comedies or action movies.

Finding and recognizing such patterns can help reduce the complexity in the data to reduce noise and bring out important trends. And also to compress it. We will see some important methods for finding such patterns, and then, some times, we do not even have feature vectors. Think back about the social network of monks that Sampson recorded back in the 60s.

The monks are our data points and we want to find groups of friends among them. Similarly, of course, for much larger networks like Facebook. Can we use k-means to find groups here? It's going to be hard. We don't have any feature vectors. All we know is who talks to whom.

So we have points, the monks, and we have relationships, who talks to whom. This gives a graph. We can draw all the monks on paper and draw lines between those who talk. The structure of points and lines is a graph. The lines are called edges. This graph tells us a whole lot about the patterns in our data.

Groups of friends will have lots of edges between them, and between such groups there are not many edges. We will see how a little bit of math will magically find us the groups in the graph. In fact, for all of the problems I mentioned, on a high level, the same approach will work.

We are going to create new features that represent our data well. These new features reveal a lot about the hidden structure in the data. Each data point can be represented bu those new features. Where do they come from? We construct them by looking at the entire data. Sounds like magic?

We will soon see how the magic works.

1.15 Finding the Principal Components in Data and Applications

We will now look at one of the most widely used methods for finding major patterns in data, principal component analysis or PCA. It is most often used when each data point contains a lot of measurements and not all of those may be meaningful or there is a lot of covariance in the measurements.

In the last section, we came across some examples, genomic data or images. PCA describes data by summarizing it in typical patterns, these are the principle components. Let's look at a concrete toy example that illustrates what kind of information we may extract. Here, we see a matrix of ratings for holiday destinations.

Each of your friends has given a rating to four holiday places. We have first, a spa hotel in the Alps. Second, a Hawaiian beach. Third, a tracking tour in the Himalayas. And fourth, a deep sea scuba diving class. Anne really likes the spa in the Alps. Bill is up for Hawaii, while Maggie loves the Himalayas.

Can we better understand this data? Are there underlying patterns? Each person here is a data point and their features are the ratings. We want to find patterns such that each person's rating can be explained as a combination of patterns. These patterns are the principal components. Each principal component is a vector.

Here the two principal components are in order of importance and slightly rounded. Component one is -1, +1, -1, +1 and component 2 is +1, +1, -1, -1. Do these patterns tell us anything? Pattern 1 has high ratings for trekking and scuba and low ratings for beach and spa.

We could say pattern one corresponds to the amount of adventure for each place. If I like adventure, pattern one tells my likings. Pattern 2 prefers Alps and Himalayas over beach and scuba. This pattern expresses a liking for mountains. Now each person's ratings can be expressed via the two patterns and a mean rating.

Anne's ratings, for example, are approximately the mean rating minus 6.7 times the adventure pattern, plus 2.2 times mountains. She has a negative coefficient for adventure. She tends to rate adventurous things low. And she likes mountains over sea. Indeed, she rates the Alpine spa the highest. We can do this with each person and the coefficients for each pattern are telling.

In fact these two patterns pretty much explains people's ratings here. Instead of four features, one rating for each location, we now have only two, adventure and mountains. We can now also visualize the data. For each person we compute the two coefficients, one for each pattern, and plot each person as a data point on these two axis.

Such visualizations can be tremendously helpful. Here natural clusters emerge. Maggie and Jenn are really adventurous whereas Chris and Bill want to relax at the beach. We just saw an example of principal component analysis or PCA. It finds the major axis of variation in the data. These are our patterns.

Each data point can be expressed as a linear combination of these patterns or components. Let's briefly look at two other famous real world examples. First components of face images. Retreat each image as a vector. We take our pixelized columns and stick them on top of each other. We can find a blurry average face.

Equally we can find a few principal components. We see each component looks like a ghostly face. These components have been termed Eigenfaces. The eigen comes from the word eigenvector as we will see in the next section. Each face image is then an overlay of weighted versions of these components.

For each image we only need to know its coefficients. This is useful for compression, but also for understanding the space of face images. Our final example is a study from genetics. It asks can you see someone's origin from their DNA? Researches collected data from roughly 1400 Europeans. Each person is described by his or her genetic variations, that is 200 000 features.

The researchers found the two principle components of the data and plotted each person, just like we did with our ratings. In the plot the data points form clusters. In these clusters we can see the map of Europe. The two principle components came only from genomic data, and still the map emerges.

We see that, indeed, the principle genetic variations in Europe are highly related to geography. PCA indicated that for us. Let us recap. The principle components capture major patterns, and each data point can be expressed via these components. We have created new features. Typically a few components are enough.

If each components describes a feature, we have reduced the number of features. This is called dimensionality reduction. Next, we will see how to compute the principal components.

1.16 The Magic of Eigenvectors I

In the last section we saw the usefulness of principal component analysis, or PCA. Now we learn how we can actually compute it. The magic is called eigenvectors, and indeed, we will see many more benefits of eigenvectors. In PCA, we want to find principal patterns. Typically, these patterns do not affect a single feature.

Or in the holiday example, a single rating. They affect many ratings together. Ratings for several holiday locations go up and down together, governed by some latent factor like adventure. This going up and down together is the covariance of the rates. They covary from the mean. We want to find the major directions of covariance.

To find those, a helpful tool is a covariance matrix of the data. Let's see how we construct it. For each holiday location, we compute the average rating. We then look how for each person the rating is deviating from that average. For example, the deviation of Hawaii for Anne is, Anne's rating for Hawaii minus the mean-rating for Hawaii.

The covariance of two locations, say Hawaii and the Himalayas, measures how much the ratings both vary in the same direction. For each person, we multiply the deviation for Hawaii and for Himalayas. If both ratings are larger than their mean, then the product is positive. If they vary from the mean in different directions, the product is negative.

We do that for all people. The covariance of Hawaii and the Himalayas, is then the sum of those. The covariance matrix contains these terms for all possible pairs of features. For D features, this is a D by D matrix, and the IJ entry is the covariance of features I and J.

In our matrix, for example, the relaxing alpine location has negative covariance with deep sea diving. People tend to like either one or the other. The principle components that we are looking for are the eigenvectors of this covariance matrix. What are eigenvectors? We can define them by a matrix in vector products.

We can multiply every length D vector by a D by D matrix. The result is another vector. You can think of a vector as a string of numbers or as a direction or arrow in three dimensional space. When we multiply it by a matrix, the vector gets rotated and scaled.

If the vector is an eigenvector, its direction stays the same. It only gets scaled. The amount by which it gets scaled, is the corresponding eigenvalue. Each eigenvector has an eigenvalue. In some sense, the eigenvectors capture the major directions that are inherent in the matrix. And the larger the eigenvalue, the more important is the vector.

This can be written as an equation. We won't solve that. The computer can do that for us. Let's, rather, try to understand what this means for covariance matrices. Here is another dataset. The data are points in 2D on a plane. Each data point has two features. In this data, either both features are low or both are high.

In fact, there may be an exact relationship and the rest is noise. The principle components here literally capture the major axis of variation. The first and most important eigenvector points in the direction in which the ellipse of the data extends most. That is the vector with the largest eigenvalue.

The eigenvalue tells the amount of stretch. That's also the major relationship between the features. We found this automatically. The second eigenvector is perpendicular and given that constraint, shows the next largest direction. This is what it looks like for another dataset with clusters. If we encode every vector only by the first principle component, we shrink all the data onto this component.

But even though everything is now on one line, the clusters are still separated. We did not lose much. All of this can be done in high damages too. This last example actually points to another question. How many components do we need? Some of them may be useful, and the rest just noise.

For that, we can look at the eigenvalues. If we sort them, there's often a gap after which all are small. We pick all the eigenvectors that correspond to large ones and ignore the rest. In summary, we've seen that we can compute the PCA from the covariance matrix. The eigenvectors with the largest eigenvalues are the principle components.

PCI is very useful for a wide range of settings. We've seen a few. It can also be used to reduce the large feature vectors that come from deep neural networks for example. Of course it can't do everything. For example, it can only catch a linear relationship in the data.

Still, it is one of the most widely used data analysis tools.

1.17 Clustering in Graphs and Networks

So far, in our data, each data point was described by a set of features. Now, we are no longer have that luxury. Take a social network. We may know nothing about the people in there, so we have no feature vectors. But we know who talks to whom, that is, who is connected.

In the Internet, two websites are connected if one links to the other. Then there are biological networks, computer networks, and data represented as networks such as images. So our data has given us a network or a graph. Each data point, for example a person in the social network, is a node in the graph.

The connections are edges. We can draw this on paper by drawing dots for nodes and lines for edges. We may have weights on the edges. The larger the weight of an edge, the stronger is the connection between the two nodes. By small graphs on paper may still be easy to get, making sense of a huge social or biological network is much harder.

A first thing to to understand may be the community structure of a network. Other dense groups of friends for example? We may also want to partition other networks into well connected groups. We can learn a lot from cluster analysis in the graph. In social networks, the cluster structure affects how academics spread.

Cluster analysis of dynamic social networks teaches us about the formation of trends and opinions. In Biology, clusters of proteins can be indicative of functionality. And in Science, clusters of coauthor networks indicate research topics. There are lots of other examples. In fact, we'll see that we can even use graphs to cluster data that is given as feature vectors, and even in cases where fails.

This happens for example if the data forms some sheets or curves. Let's try to make this a bit more formal. What is a cluster in a graph? There are in fact many definitions. But many of them show some intuitive criteria. Let's have a look at those. Intuitively, a cluster consists of points that are well connected with each other.

That is, there a lots of edges between them. The number of edges between a cluster is also called the volume. The volume per node is the density. We want this to be large, that's the first breakthrough. Second, in different clusters there should not be many edges. After all, if there were many edges, why would you not declare this as a single cluster?

This separation of clusters is measured by the cut value. If we wanted to cut out the cluster from the graph, we would need to disconnect the edges that connect the cluster with the rest of the cloud. The cut value is the number of edges we'd need to cut.

Or the rate. So, we could just find a group of nodes with a lower cut value. Is this a good criteria? Certainly, most clusters have a low cut value, but there are clusters that have an even smaller cut. Look at this outlier node. It has only one edge so its Cut (C) value is one.

That's super low, and hence by our definition makes a good cluster. Maybe that's not exactly what we were aiming for. Instead, combining cut and volume strikes a balance between the size of the clusters and their separation. Popular examples of such combined Fertilia are conductance and normalized cut. Both divide the cut by a measure of volume.

We want to minimize this criteria. If the cluster is small and has small volume, then the denominator is small and the criteria is large. If the cluster is large, then the other cluster that is the remaining nodes is small. By this measure, good clusters are not too small internally are connected and separated from the rest of the nodes.

Clusters can also be seen in the so called adjacency matrix. We can construct a matrix that has a row and a column for each node. The entry in row i and column j is 1 if there is an edge between node i and j, and it's 0 otherwise.

If your other nodes or rows and columns by cluster then the matrix has a block structure. Typically, we do not know the clusters and the nodes are shuffled arbitrarily. So the structure is hidden in what looks like chaos. There are many other flavors of clustering. For example, modularity measure indicates how high the density of edges is within groups compared to a baseline graph where edges occur randomly.

Or sometimes if the graph is massive, you're only interested in very local clusters. For such local clustering you don't need to look at the entire graph. Finally, in correlation clustering, we have information for pairs of points. If they are similar, they should be in the same cluster. If they should be separated, we draw a negative edge between them.

We then cluster points to respect as many of these given relationships as possible. Here, we'll continue with the goal of finding dents while separating clusters. Next, we'll see how to do it.

1.18 The Magic of Eigenvectors II

As we have seen, graph clustering can be useful to understand community structure in a large network. But how do we compute that? What is important here is the global connectivity structure of the graph. To capture that connectivity structure, we will see that, once more, eigenvectors can be really useful.

They encode an impressive amount of information about the structure of the graph. We almost just need to read it out to find the clusters. The resulting methods are called spectral methods, and more more precisely, spectral clustering. The spectrum of a matrix is the set of its eigenvalues. We've seen eigenvectors for PCA, but what is the matrix for a graph that gives us the magic eigenvectors?

The matrix is the so called Laplacian of the graph. Let's construct it step by step to see what it is. Here's an example graph for illustration. We number the nodes one through n. For a graph with n nodes, the Laplacian is an n by n matrix. It has one row and one column for each node.

The entry in row i and column j corresponds to a potential edge between nodes i and j. If there's an edge, the entry is -1, otherwise it's 0. The entries on the diagonal are the degrees of the nodes. The degree of a node is the number of edges that it meets.

Often this Laplacian is normalized by appropriately dividing by the degrees or the square root of the degrees. One can also define all of that for graphs. Now what is so special about this matrix? We'll see. Let's have a look at the eigenvectors of the Laplacian. Remember, each eigenvector comes with an eigenvalue.

We'll sort them by the eigenvalues in increasing order. We will be interested in the largest and the smallest one. The smallest Eigen value is always 0. In fact, here's our first structural relation. The number of 0 eigenvalues is exactly the number of disconnected parts of the graph. These parts are called connected components.

For example, this graph has three connected components and this one has one. Let's assume our graph is one component and let's have a look at the eigenvectors. The eigenvector for eigenvalue 0 is the all once vector. That's the same for all graphs, and hence pretty uninteresting. So let's go on to the second smallest eigenvalue and vector.

Each item vector has n entries, one for each node in the graph. Here are two graphs, and we color code the notes by the value in the item vector. Now this is interesting. The values in the eigenvector reflect the graph's vector. Close by nodes have similar colors. In fact, we could find a threshold to partition the nodes.

All nodes with values above the threshold go in one group and all others in the other group. If we do this here, we actually find a meaningful clustering in the graph. This eigenvector has a lot of good information for us. What about others? The nest few eigenvectors complement that information.

Here's a graph with four natural clusters. Our second eigenvector partitions this nicely into two groups, each consisting of two clusters. The color coding for the third eigenvector also reflects clusters in the graph, but partitions the graph differently. If we take this information for both eigenvectors together, we can easily find all four clusters.

This seems extremely useful. The eigenvector is kept to the graph structure. In fact, they gave us more. Remember, our nodes do not have feature vectors. Now, we can give each node one feature value from each eigenvector and we suddenly have vectors. These new features are called an embedding.

PCI also gave an embedding. For PCI, we use the vectors with the largest eigenvalues. Here, the smallest ones will be important. What do these features mean? Just like we plotted the fetus from PCA, let's take the second and third smallest eigenvectors and use them as coordinates for the nodes.

Here's what this looks like. Nodes that are connected are positioned close by on the plane. Our embedding here wants to minimize the stretch of the edges. And clusters in the graph become clusters in 2D. So far, we've used the eigenvectors with lowest eigenvalues. What about the ones with the largest eigenvalues?

Here is what happens if we use these as an embedding. They have the opposite effect. They maximize the stretch of the edges. Nodes that are connected are placed far away on the plane. This is the same graph, but looks so different. Is there an explanation? In fact, the eigenvectors minimize or maximize a different score.

Remember, each node gets a value. For each edge, we take the difference of the adjacent node values and square that. We do this for all edges and sum it up. The eigenvectors with small eigenvalues minimize these differences. They give similar values to connected nodes. And the difference can be viewed as the stretch of the edge.

The eigenvectors with large eigenvalues maximize the difference, they get very different values to connect the nodes. Overall, we see that the eigenvectors and eigenvalues contain a lot of valuable information about the graph structure, as we may suspect now, they are very useful for clustering. In fact, they contain even more information about the number of paths between any two nodes or about the importance of nodes and edges.

1.19 Spectral Clustering

Communities and networks give us a much better understanding of network properties, friendships or functional groups. In the past sections, we've seen criteria for finding communities. And we've seen that Eigenvectors kept your important properties of the network. Now we put everything together. Let's start with a graph where there are no connections between communities, and build the so-called adjacency matrix.

For every node, we put the strength of the connection, or edge, to every other node in the matrix. If there's no connection, we'll put a 0. The Laplacian matrix that we've seen previously is closely related. Put the row sums on the diagonal and negate the other numbers. Both matrices have the same block structure, and each block consists of edges of one community.

Of course, in a real network, we don't know how to order the nodes to make this block pattern emerge. So, how do we find the blocks? Let's take a look at the Eigenvectors of the matrix. The bottom three vectors, exactly indicate the block structure. And they each exactly indicate the nodes of one community.

The remaining eigenvectors are less important here. This works even if the nodes are scrambled. For each node, we create a feature vector as we seen before. One entry from each eigenvector. These feature vectors directly indicate which community the node belongs to. In reality of course, the communities are not separated so clearly.

There are edges between communities and they just within community are missing. But if there are not too many of these extra edges, the eigenvectors don't change too much and things still work out. To see this, we can use our new features as an embedding. Just like we've done previously.

Here we have three features so three dimensions. We plot each node on these three axis and seen that the graph communities emerged as well as separated clusters. This suggests the following approach called spectral clustering. First, we write down the Laplacian which comes from the adjacency matrix. Then, we compute the eigenvectors and keep the few bottom ones.

The eigenvectors give new features. We use k-means clustering on those new features. This works because the new features magically separate the communities. To find k clusters, one typically uses K eigenvectors. If the graph is completely connected, there is pouring eigenvector of all ones that we discarded right away.

There are variant of spectral clustering but the main idea is the same. Now, recall our criteria from the very beginning. Where we wanted clusters of decent size with small cut between them.

This sounds very different from our eigenvectors here but in fact, there's a close connection. We can give labels to notes to indicate the optimal clusters for the criteria.

For example, a special positive number for cluster one, and a special negative number for cluster two. The eigenvectors approximate those labels. Actually, spectral clustering is widely used even for data that is not a network. Recall k means. If we plot the data points, the clusters we find with k means are round.

So here's an example of data with clusters but they are not round. The data seems to curve around and intuitively, we'd like to follow that curvy structure. We run k-means and the clusters look a bit odd. They disregard the shape of the data. Well, spectral clustering can help you.

We translate the data into the world of graphs. Concretely, we build a graph from the data. For each data point, we create a node and connect it to its closest or most similar points nearby. The graph now reflects local relationships between our data points. It follows the shape of the data.

Now we use spectral clustering. We get clusters of points that are densely connected in the graph. These clusters do follow the shape. They look very different and more meaningful. In summary, to use spectral clustering on other data, we just add one step to our procedure. Create the so-called neighborhood graph.

A bit of care is needed when building this graph. An edge between two points is weighted by the similarity of the points. A common similarity function is the Gaussian similarity that decays exponentially with the distance. They connect each point to a fixed number of closest neighboring points. How many?

Typically, it should not be too many. But enough that the resulting graph has no or very few disconnected parts. What is the difference between spectral clustering and k-means here? The interpoint distance is used by k-means don't capture the complicated shape of the data. In some sense, we don't want to jump across the gaps.

A graph catches those gaps. Another interesting point, our spectral clustering also uses k-means but with an important difference. We created new feature vectors and used k-means with those new features. Then, the clusters follow the inherent shape of the data. In other words, we found an embedding for our data points that captures the latent structure much better.

We've seen embeddings with PCA, but this one is different. In PCA, the new features are some combination of the existing features. Even though we didn't explicitly write it that way. With a graph, there is no such relatively simple relation. Here we have a non-linear embedding. In summary, by putting everything together, we saw how to use the eigenvectors off the Laplacian matrix, to find meaningful clusters that respect hidden structure in the data.

1.20 Modularity Clustering

In the previous section, we have seen how we can explicitly use eigenvectors to recover hidden communities in a graph and in fairly complex data in general. Next, we will explore another very related criterion for finding communities. There is one difference, in the previous approach, we needed to specify the number of communities.

Our next method does that automatically. This method is called modularity clustering, and has been viably used to analyze all kinds of networks. For this this method, we need to define a modularity score that we would like to maximize. This score applies to a given clustering into communities. The characteristic idea of modularity is to compare the communities to a random baseline graph that shares some properties with our actual graph, such as the number of edges and roughly the degree of the nodes.

Good communities should have many more edges inside than expected by the baseline. So, let's say we have two candidate communities, how do we measure their deviation from a baseline? We compute the number of edges within the community and then subtract the expected number of edges as per the baseline model.

Concretely, for every pair of nodes i and j that are in the same community, we count aij minus pij. Aij indicates an edge, it is 1 if i and j are connected and 0 otherwise. Pij is the expected number of edges between i and j in the baseline graph.

We sum this over all pairs. For each community, we get the actual total number of edges in the community minus the expected number of edges within the community. If that is large, then the community is surprisingly dense. This score can be generalize to a larger number of communities.

What is the expected baseline number of edges? There are several ways to define that baseline model, here's a common one. The expected number of edges between node i and node j is proportional to the degree of i times the degree of j. So, nodes with many edges in our graph are expected to have many edges in the baseline, too.

The idea of modularity is that an arbitrary set of nodes, say people in a social network, may have just as many connections as the baseline expect. In that case, the modularity is close to zero. In the example here, we have arbitrary partition into meaningless communities. The modularity of this partition is low, but if we pick exceptionally well connected communities, then the modularity is high.

In fact, modularity can be -2. This happens if the number of edges in our community is much lower than expected. It may look like this example, this looks like our edge stretch plots with eigenvectors. That's not a coincidence, the phenomenon is closely related. Such communities can be useful, too.

Say the nodes in this graph are words, and, they have an edge if they co-occur in a collection of texts. Verbs and nouns typically co-occur, but not two verbs or two nouns, so this partitioning would separate verbs and nouns, but let's get back to dense communities and find communities that maximize modularity.

Note that we do not need to specify the number of communities. We pick whichever number is best for the score, it happens automatically. How do we find a good partitioning? We could try all possibilities, but that's going to take really long, even with a very fast computer. One approach is to start with some partitioning and then move nodes between groups if that improves the score, but it turns out we can use eigenvectors.

This time we use a matrix that is related to the adjacency matrix and the Laplacian we have already seen, but it's slightly different. Remember, our sum over Aij minus Pij to compute the modularity. Our matrix has one entry for each pair of nodes, and the entry in row i and column j is exactly Aij minus Pij.

Remember, Aij indicates if there's an edge, and Pij is the expected number of edges between i and j. We compute the eigenvectors of this matrix, again, each eigenvector assigns a value to each node. For two communities, we use the eigenvector with the largest eigenvalue. We put all nodes with positive values in one community, and all nodes with negative values in the other.

We can extend this to more eigenvectors and communities. The actual value given by the eigenvector shows how important or central a node is in the community. Now there's one important difference to our previous vector Laplacian, the eigenvalues here can be negative. A negative eigenvalue means that the associated eigenvector gives a node value that leads to low modularity.

Let's conclude with a famous example, Zachary's karate club network. In this network here, each node is a member of a karate club at a US university in the 1970s. The edges indicate friendships between the members, using modularity we find two communities in this network. Now the interesting fact is that shortly after the data collection, there was an internal dispute, and the karate club split into two groups.

And guess what, the communities found by modularity clustering are exactly the two groups that resided from the split.

1.21 Embeddings and Components

In the previous sections, we've seen PCAs, spectral embeddings, and spectral clustering. All these methods find new feature vectors for the data points, in other words, an embedding. The idea of embeddings is very useful. So, let's summarize some different types of embeddings and their uses. With eigenvectors, we already saw that different embeddings can look very different.

We used embeddings to do dimensionality reduction. We found short future vectors, since finding important information in high dimensions is like finding a needle in a haystack. Our two dimensional embeddings visualize graphs, news, or people's preferences. The new feature vectors bring out major trends, so it's easier to apply other methods.

In spectral clustering, k-means give better clusterings after spectral embedding. It's very easy to separate the clusters now. And embeddings can give us important components or themes, as with PCA. This points to a difference between the spectral graph embeddings in PCA. In PCA, each data point could be written as a way to culmination of the components, such embeddings are called linear.

In spectral clustering, we first construct the graph and then find eigenvectors. Even if we start out with future vectors, our new representation is not linear, such embeddings are non-linear. The different types of embeddings are guided by different goals and criteria. In spectral clustering, all of the new appearance similarities, like friendships in a social network or the related articles for given news articles in a collection.

Our goal is to find a vector representation that maintains these local relationships and puts related points close to each other. This is what we saw in our visualizations. This goal of low dimensional features that preserves some pair relationships is shared by other methods. For example, sometimes we have side information available to guide the embedding.

Say, we have images of handwritten digits and we know the correct digits for some of them. We like an embedding that places all 2s together, all 3s together and so on. We can guide our embedding by giving it example pairs that should be close because we know they are the same digit and example pairs that should not be close because those images are different digits.

Apart from that, we want to preserve local relationships. Metric learning is a method that takes such side information to guide the embedding. In methods like PCA, we are not directly looking for a clustering but for prominent patterns in our data. And each data point may be associated with more than one pattern.

The idea of finding prevalent patterns, components, topics, or themes appears in several other methods. Independent component analysis, or ICA, recovers components that are statistically independent. That is, they have no information about each other. This is stronger than the difference between the PCA components. Think of a microphone that records an overlay of noise from a dishwasher, a barking dog and people speaking.

ICA can be used to recover all these different sources of sounds. Dictionary learning poses constraints on the data point associations. Each data point may be associated with only very few components. In this case, we may need many components. So we may not reduce the dimensionality. But we obtain a dictionary that can be used, for example, to reconstruct corrupted images.

Let's go into a bit more detail for one of the methods that is closely related to PCA. Singular Value Decomposition, or SVD. In fact, it also gives us the principle components. Recall our PCA example, our friends' ratings of holiday places. Each row in our data matrix corresponds to one person and each column to a place.

Each principle component is a pattern, like adventure or mountains. And associates the places with those patterns. SVD will give us the associations of both places and people for each pattern. How does it work? We take our data matrix, and as for PCA, subtract the means. This gives us the matrix x.

The covariance matrix in PCA is x times x transpose, but here, we keep x. The singular value decomposition writes our data matrix x as a product of three matrices. These are usually called u, sigma and V transpose. The matrix sigma is zero, except for the diagonal. The squares of the diagonal entries in sigma are the eigenvalues we get in PCA.

The roles of these principles are the same as the eigenvectors in PCA. These are called the right singular vectors. The columns of the matrices are the left singular vectors. These three matrices describe our data via themes. For each theme, we have a singular value and a left and right singular vector.

The singular value tells the importance of the theme. Higher is more important. The right singular vector shows the association of each place with the theme. Recall, scuba diving was associated with adventure but not mountains. The corresponding left singular vector tells the preference of each person for the particular pattern.

So, we can simultaneously understand the likings of people and the characteristics of the places. The entries in the left vectors, multiplied by their respective singular values, are exactly the coordinate we use to visualize the data. The same kind of analysis is useful when the rows of our data are customers and the columns are products.

Or, if we describe drug interactions, the rows may be drugs, and the columns are proteins or pathways in the body. So, let's summarize. Embeddings give us meaningful dimensionality reductions, bring out hidden clusters in the data, and help us understand the data via major patterns, components or themes.

1.22 Case Study 1.2: Spectral Clustering: Grouping News Stories

Now, let's apply spectral clustering to a real-world example, news stories. News websites like Google News group articles by topics. Let's see how we can construct a grouping like that. We collected 60 news articles from the Guardian, Independent, BBC and CNN. On those websites the articles had the tags Brexit, Trump, USA, Middle East, Africa, Euro 2016 and cricket.

For our analysis we ignore the tags. We only use the words on the articles. First you build a feature vector for each article. Then we compute a similarity for each node as an article. Then we complete the Laplacian Eigenvectors, and this will give us a visualization and a clustering.

So let's start by creating a vector for each article that describes it. Here's one example article. This article was published in The Guardian right after the Brexit referendum in the UK, on the 23rd of June, 2016. The article has 720 words. We use a library to extract all entities mentioned in the article.

Names, organizations, and locations. Here are the entities detected in the article and how often they occur. Bank of England and BBC occur once. Brexit occurs three times, Britain six times, and so on. How can we use this to compute similarities between articles? We collect all the entity words from all the articles in the dictionary.

These are 1,063 words in total. Here are some of the words in our dictionary. Each word is assigned an ID number between zero and 1,062. Now each article can be represented as a vector of 1,063 counts, one for each entity. For example, the word written has ID 138, and it occurs six times in our article.

So the count vector of our article has six in position 138. To make these counts more meaningful, we convert them to the TFIDF format. This stands for Term-frequency Inverse Document Frequency, and indicates the importance of a word. This value increases with the number of times the word occurs in an article, but decreases if the word occurs often in many articles.

The term frequency is the number of times a word occurs in the article. That is our count divided by the number in entities in this article. So if Britain occurs six times in and article with 200 entities then the term frequency would be six divided by 200. We divide this by the number of articles this word occurs in, for example 35.

We use our tf-idf vectors to compute similarities between all articles. For any two articles we sum up the square differences of their tf-idf vectors and exponentiate that. Now we build a graph of articles by connecting each article to its ten closest articles, then we apply spectral clustering. We get seven clusters.

Cluster one contains for example the articles Can Donald Trump afford to be president and Bernie Sanders I will vote for Hillary Clinton to stop Donald Trump. Cluster three contains, for example, the Brexit article Merkel says no need to be nasty in leaving talks. Indeed, the clusters correspond pretty much to the tags given by the news websites.

Here we see how the articles of each tag are clustered. One color for each cluster. The tags that are put together here are USA and Trump. And indeed, these articles contain very similar words, but we get much more. Let's use the Eigenvectors of the second and third smallest eigen values to plot our data.

Each article now has two coordinates, the entry in each of these Eigenvectors. Each dot is an article. The colors indicate the predicted clusters. And here, we label some articles by their tags from the news websites. This plot shows the relationships between articles. Even without the cluster labelings, we see different directions emerging.

The top left are articles on cricket, and the bottom left are articles on Brexit. Towards the right we get a stretch of articles on USA and Trump. Let's have a look at the purple group on the right. It contains articles from Middle East and Africa, for example one talks about an attack on a hotel in Somalia and one about Russian air strikes on Syria.

These are articles about war and attacks. One Middle East article is positioned much closer to the center and the Europe. It's title is The orchestra of Syrian musicians: 'when there is violence, you have to make music'. It talks about musicians from a former Syrian national orchestra who were refuges and were reunited to go on a Europe tour.

This article also mentions how the conductor is still waiting in the US. It mentions the public opinion towards refugees and the Brexit. In short, it touches upon many of the other topics, so it's drawn to the middle of the plot. Here, we already see some clusters emerging. But some clusters are still somewhat close together.

They get separated when we use other Eigenvectors. In summary, we've seen how a spectral embedding arranges news articles by content and finds topic groups. It reveals whether there is any group structure in our data. And it illustrates relationships in the data.

Module – 2: Regression and Prediction(Victor Chernozhukov)

2.1 Classical Linear & Non-Linear Regression & Extension

2.1.1 Introduction

Hi, I'm Victor Chernozhukov, professor in the Department of Economics and Center for Statistics at MIT. My research in teaching for this econometrics and mathematical statistics was much of recent efforts due added to the use of machine learning methods for causal Inference. In this module, we will discuss regression in classical and modern versions.

Regression analysis is about discovering correlations between the outcome, y and the set of regressors x, which sometimes are also called features. In this module, y is always the real random variable and x is the vector of random variables, with components denoted by X1 through Xp. Where we always assume that a concept of one is included as one of the components.

We denote the transpose of a vector by the prime sign. For example, suppose our y is hourly wage which varies across workers, and x is the retro regressors which include variables that measure education, experience gender, and other job relevant characteristics. When we are interested in the effect of a particular component of x and y, we show partition axis for D and W, where D will be the target regressor and W will be called the controls of components.

The purpose of the regression analysis is to characterize the statistical relation y to x. Namely, we want to answer two question. Question one, or the prediction question. How can we use x to predict y well? Question two, or the inference question. How does the predicted value of y change if we change a component of x, holding the rest of the components of x fixed?

In our wage example, the predictions questions becomes how to use job relevant characteristics, such as education, experience, gender and others, to best predict the wage? An inference question becomes for example, how gender affects the predicted wage? This allows us to uncover gender discrimination if it exists. In this module, we will discuss classical and modern regression tools that answer these two questions.

We'll now give a preview of some of the empirical findings from a case study in which we will examine our wage example using data from the US current population survey in 2012 for single workers. We will address the two questions post. Number one, we will construct a prediction rule for a hourly wage which depends on the job relevant characteristics linearly.

We will assess the quality of the prediction rule using our sample prediction performance. Number two, we will find that an average women get about $2 less an hourly wage than men with the same experience and other recorded characteristics. This is called the gender wage gap in labor economics and measures in part gender paid discrimination

2.1.2 Linear Regression for Prediction

We'll start out by looking at the linear regression problem at the population level. This means that we have access to infinite amounts of data, and therefore we can compute theoretical expected value of population averages such as expected value, or expected value of Y times X. Later for estimation purposes, we will have to work with a finer sample of observations drawn from the population.

But right now we focus on the population case to define the ideal quantities. We shall try to construct the best linear prediction rule for Y using a vector of X, with components denoted by Xj. Specifically given such X, our predicted value of Y will be beta prime X, which is defined as the sum of beta js times Xjs.

Here the primary vector beta consists of components beta js, and we commonly call this parameter the Regression Parameter. We define beta as any solution to the Best Linear Prediction problem, abbreviated as BLP in the population. Here we minimize the expected or mean squared error that results from predicting Y, using linear rule given by b prime X.

Where b denotes a potential value for the parameter vector beta. The solution, beta prime X, is called the Best Linear Predictor of Y using X. In words, this solution is the best linear prediction rule among all the possible linear prediction rules. Next we can compute an optimal beta by solving the first order conditions of the BLP problem, called the Normal Equations, where we have expected value of (Y- beta prime X) = 0.

Here we are setting to 0 the derivative of the objective function with respect to b that will minimize. Any optimal value of b satisfies the normal equations, and hence we can set beta to be any solution. Under some conditions such solution will be unique, but we don't require this in our analysis.

Next, if we define the Regression Error as epsilon = (Y- beta X), then the normal equations in the population become expectation of epsilon times X = 0. This immediately gives us the decomposition Y = beta X + epsilon, where epsilon is uncorrelated to X. Thus beta prime X is the predicted or explained part of Y, and epsilon is the residual or unexplained part.

In practice, we don't have access to the population data. Instead we observe a sample of observations of size n, where observations are denoted by Yi and Xi, and i runs from 1 to n. We assume that observations are generated as a random sample, from a distribution F, which is the population distribution of R, Y, and X.

Formally this means that the observations are obtained as realizations of independently and identically distributed copies of the random vector Y, X. We next construct the best linear prediction rule in sample for Y using X. Specifically, given X, our predicated value of Y will be hat beta prime X, which is the sum of hat beta js times Xjs.

Here hat beta is a vector with components denoted by hat beta js. We call these components the Sample Regression Coefficients. Next we define the linear regression in the sample by analogy to the population problem. Specifically we define hat beta as any solution to the best linear prediction problem in the sample.

Where we minimize the sample mean squared error for predicting Y using the linear rule b prime X. Here we denote the sample mean, or empirical expectation, by the symbol E sub n, which simply is the sample average notation. In words, the empirical expectation is simply the sample analog of the population expectation.

We can compute an optimal beta hat by solving the sample normal equations, where we set the empirical expectation of Xi(Yi- beta hat prime Xi) = 0. These equations are the First Order Conditions of the Best Linear Predictor problem in the sample. By defining the In-Sample Regression Error, hat epsilon, as (Yi- hat beta prime Xi), we obtained the decomposition of Yi into sum of two parts.

Part one, given by hat beta prime Xi, represents the predicted or explained part of Yi. Part two, given by hat epsilon i, represents the residual or unexplained part of Yi. Next we examine the quality of prediction that the sample linear regression provides. We know that the best linear predictor of our sample is beta X.

So the question really is, does the sample best linear predictor hat beta X adequately approximate the best linear predictor beta X? So let's think about it. Sample linear regression estimates P parameters, beta 1 through beta p, without imposing any restrictions on these parameters. And so intuitively, to estimate each of these parameters, we will need many observations per each such parameter.

This means that the ratio of N / P must be large, or P / N must be small. This intuition is indeed supported by the following theoretical result, which reads. Under regularity conditions, the root of the expected square difference between the best linear predictor, and the sample best linear predictor, is bounded above by a constant, times the level of noise, times square root of the dimension p / n.

Here we are averaging over values of X, and the bound holds with probability close to 1 for large enough sample sizes. The bound mainly reflects the estimation error in hat beta, since we are averaging over the values of X. In other words, if n is large and p is much smaller than n, for nearly all realizations of the sample, the sample linear regression gets really close to the population linear regression.

So let us summarize. First, we define linear regression in the population and in the sample through the best linear prediction problems solved in the population and in the sample. Second, we argued that the sample linear regression, or best linear predictor, approximates the population linear regression, or best linear predictor in the population very well when the ratio p / n is small.

In the next segment we will discuss the assessment of prediction performance in practice.

2.1.3 Assessment of Prediction Quality

In this segment, we have two learning goals. Goal 1 is to understand the analysis of variance, or ANOVA, in the population and in the sample. Goal 2 is to assess the out of sample predictive performance of the sample linear regression. We begin with the population case. Using the decomposition of Y into the explain and the residual part.

And the normal equations that we derived in the previous segment, we can decompose variation of Y into the sum of explain variation and residual variation, as shown in the formula. We next define the population mean squared prediction error, or MSE, as the variance of the population residual. The expectation of epsilon squared.

We also define the population R squared as the ratio of explained variation to the total variation. In other words, the population R squared is the variation of Y explained by the BLP. And as such, it is bounded below by 0 and above by 1. The analysis of variants in the sample process analogous.

We simply replace population expectations by the empirical expectations. Using the decomposition of Yi to explain and the residual part, and the normal equations for the sample. We can decompose the sample variation of Yi into the sum of explained variation and residual variation. The former is given by the sample variation of this sample best in a predictor.

And the latter is given by the sample variation of the residual. We can define the sample MSE as the sample variance of the residuals and we can define the sample R squared as the ratio of the X-plane to the total variation in the sample. We know that when P/N is small, the sample linear predictor gets really close to the best linear predictor.

Thus when P/N is small, we expect that sample averages of y squared beta hat xi squared epsilon hat i squared to be close to the population averages of Y squared, beta X squared, and epsilon squared. So in this case, the sample R squared and the sample MSE will be close to the true quantities, the population R squared and MSE.

When P / N is not small, the discrepancy between the two sets of measures can be substantial. And the sample R squared and the sample MSE can be very poor measures of predictability. For example, when p = n, we can have sample MSE = 0 and sample R squared = 1 no matter what the population MSE or R squared are.

The following simulation example will support our reason. In this example, Y and X are statistically independent and generated from the normal distributions with mean need 0 and variance 1. The means that the true linear predictor of Y given X is simply 0 and the true R squared is also 0.

Suppose the number of observations is N, the number of regressors is P. If p = n, then the typical sample R squared will be 1. Which is very far away from the true number of 0. This is an example of extreme over fitting. If p= n / 2 then the typical sample r squared is about half which is still far off from the truth.

If p = n / 20, then the typical sample r squared is about 0.05 which is no longer far off from the truth. We can now see that using sample R squared and MSE to judge predicted performance could be misleading when P over N is not small. So the question is, can we design better metrics for predicted performance, and the answer is yes.

The first proposal is to use the adjusted R squared and MSE. We first define the adjusted MSE by rescaling the sample MSE by a factor of n over n- p. And then making the corresponding adjustment to the sample R squared. The re-scaling will correct for over-fitting but under rather strong assumptions.

A more universal way to measure predictive performance is to perform data splitting. First, we use a random part of data, say half of data, for estimating or training the prediction rules. Second, we use the other part of data to evaluate the predictive performance or the rule, recording the out-of-sample MSE or R squared.

Accordingly we call the first part of data, the training sample. And the second part, the testing or validation sample. Indeed suppose we use n observations for training and m for testing or validation. Let capital V denote the names of observations in the test sample. Then the out of sample or test means squared error is defined as the average squared prediction error, where we predict yk in the test sample by head beta xk.

Where hat beta was computed on the training sample. The out of sample R squared is defined accordingly as 1- the ratio of the test MSE to the variation of the outcome in the test sample. So let us summarize. First we discussed the analysis of variance, in population and in the sample.

Second, we define good measures of predictive performance, based on adjusted MSE and R squared, and based on data splitting. And next we will discuss a real application to predicting wages.

2.1.4 Case Study: Predicting Wage 1

In this segment, we will examine a real example where we will predict worker's wages using a linear combination of characteristics. And we will assess the predictive performance of our prediction rules using the adjusted MSE and R squared, as well as out of sample MSE and R squared.

Our data for this example comes from the March supplement of U.S. Current Population Survey for the year 2012. We focus on the single workers with education level equal to high school, some college, or college graduates. The sample size is about 4,000. Our outcome variable Y is hourly wage.

And our X are various characteristics of workers, such as gender, experience, education, and geographical indicators. The following table shows some descriptive statistics. From the table, we see that the average wage is about $15 per hour. 42% of workers are women. Average experience is 13 years. 38% of college graduates, 32% have done some college work, and 30% hold only high school diploma.

You can also see geographical distribution of workers across major geographical regions of the United States. We consider two predictive models, Basic and Flexible. In the Basic Model, X consists of the female indicator D, and other controls W, which include the constant, experience, experience squared, experience cubed, education, and regional indicators.

The basic model has 10 regressors in total. In the flexible model, regressors consist of all the regressors in the basic model, plus, their two-way interactions or products. An example of a regressor created by a two-way interaction is the experience variable times the indicator of having a college degree.

Another example is indicator of having a high school diploma times the indicator of working in the Northeast region. The flexible model has 33 regressors. Given that P over N is quite small here, the sample linear regression should approximate the population linear regression quite well. Accordingly, we expect the sample R squared to agree with adjusted R squared, and they should both provide a good measure of out-of-sample performance.

The following table shows the results for the basic and the flexible linear regressions. The sample and adjusted R squared close to each other for both basic and flexible models. We also see that the predictive performance of the basic and flexible regression models is quite similar with adjusted MSEs and R squared not being very different from each other.

The flexible model is performing just a tiny bit better, having slightly higher adjusted R squared and slightly lower adjusted MSE. Next, we deport the out of sample predicted performance measured by the test MSE and test R squared. Here, we report the results for of the data into the training and testing sample.

The numbers reported actually vary across different data splits, so it is a good idea to average the results over several data splits. By looking at the results for several splits, we can conclude that the basic and flexible models perform about the same. In this segment, using a real example, we have assessed predictive performance of two linear prediction rules.

They both performed similarly, with the flexible rule performing slightly better out of sample. Next, we will proceed to discuss the Inference Problem.

2.1.5 Inference for Linear Regression

In this segment, we provide an answer to the inference question. To recall the question, we partition vector of regressors X into D and W, where D represents the target regressors of interest, and W represents other regressors, which we sometimes call the controls. We write Y = the predicted value which is beta 1D + beta 2W plus noise.

And now we recall the inference question. Which is how does the predicted value of Y change if we increase D by a unit holding W fixed? For instance, in the wage example, the inference question can be stated as follows. What is the difference in the predicted wage for a man and woman with the same job-relevant characteristics?

The answer to this question is the population regression coefficient beta 1 corresponding to the target regressor D. In the wage example, D is the female indicator and beta 1 is called the gender wage gap. Next we attempt to understand better the meaning of beta 1 using a tool called partialling-out.

In the population, we define the partialling-out operation as a procedure that takes a random variable V and creates a residual to the V by subtracting the part of V that is linearly predicted by W. So we can write tilde V as V- gamma V where gamma indexed by vw solves the problem of best linear prediction of V using W.

Here we use notation argument that does not solution of the minimization problem. It can be shown that the partialling-out operation is linear. So if we have Y = V + U, then tilde Y = tilde V + tilde U. We now apply partialling-out to both sides of our regression equation to get the partialled-out version which implies that we obtain the decomposition which reads tilde Y = beta 1 tilde D + epsilon, where epsilon is uncorrelated with tilde D.

This decomposition follows because partialling out eliminates all the Ws, and also leaves epsilon untouched, since epsilon is linearly unpredictable by X, and therefore by W by definition. Moreover, since tilde D is a linear function of X, epsilon and tilde D are not correlated in our decomposition. Now recognize our decomposition as the normal equations for the population regression of tilde Y and tilde D.

That is, we've just obtained the Frisch-Waugh-Lovell theorem, which states that the population linear regression coefficient beta 1 can be recovered from the population linear regression of tilde Y on tilde D. Beta 1 is a solution to the best linear prediction problem, where we predict tilde Y by a linear function of tilde D.

We can also give an explicit formula for beta 1, which is given by the ratio of two averages that you see in this formula. We also see that beta 1 is uniquely defined if D is not perfectly predicted by the Ws. So the tilde D has a non-zero variance.

The Frisch-Waugh-Lovell theorem is a remarkable fact which is useful for both interpretation and estimation. It asserts that beta 1 can be interpreted as a regression coefficient of residualized Y on residualized D, where the residuals are defined by taking out or partialling-out the linear effect of W from Y and D.

We next proceed to discuss estimation of beta 1. In the sample, we will mimic the partialling-out in the population. When p over n is small we can rely on the sample linear regression, that is on ordinary in these squares. When p over n is not small using sample linear regression for partialling-out is not a good idea.

What we can do instead is do penalized regression or variables selection, which we will discuss later in this module. So let us assume that p over n is small. So it is appropriate to use the sample linear regression for partialling-out. Of course, by the Frisch–Waugh–Lovell theorem applied to the sample instead of the population, the sample linear regression of Y on D and W gives us an estimator hat beta 1, which is numerically equivalent to the estimator obtained via sample partialling-out.

It is still useful to give the formula for hat beta 1 in terms of sample partialling-out where we use checked quantities to denote the residuals that are left after predicting the variables in the sample with the controls. The population partialling-out is replaced here by the sample partialling-out where V replace the population expectation by the empirical expectation.

Using the formula, it can be shown that the following results calls. If p over n is small, then the estimation error in the estimated residualized quantities has a negligible effect on hat beta 1, and hat beta 1 is approximately distributed as normal variable with mean beta 1 and variance V over n where the expression of the variance appears over here.

In words, we can say that the estimator hat beta 1 concentrates in a square root of V over n neighborhood of beta 1 with deviations controlled by the normal law. We can now define the standard error for hat beta 1 as square root of hat V over n, where hat V is an estimator of V.

This result implies that the interval given by the estimate +- 2 standard errors covers the true value of beta 1, for most realizations of the data sample. Or more precisely approximately 95% of realizations of the data sample. If we replace 2 by other constants, we get other coverage probabilities.

In other words, if our data sample is not extremely unusual, the interval covers the truth. For this reason, this interval is called the confidence interval. For example, in the wage example, our estimate of gender hourly wage gap is about -2$ and then 95% confidence interval is about -1$ to -3$.

Let us summarize, first we interpreted beta 1 as the regression coefficient in the univariate regression of the response variable and the target variable, after we have removed the linear effect of other variables. Second, we noted that this result is useful for interpretation and understanding of the regression coefficient.

This result will also be super useful for setting up inference in modern high-dimensional settings which we will discussed later in this module. And next, we will carry out a case study for our wage example.

2.1.6 Case Study: Gender Wage Gap

In this segment, we consider the case study of the gender wage gap in the United States. Here we ask and answer the following question. What is the difference in predicted wages between men and women with the same job-relevant characteristics? We work with the U.S. current population survey data for the year 2012.

We first take a look at the following descriptive statistics for the subsamples of single men and single women with education attainment equal to high school, some college, or college. The min hourly wage is \$16 for men, and about \$15 for women. So the difference is \$1, without controlling for job relevant characteristics.

If we take a look at some of these characteristics, we see that on average, men have more experience, but women are more likely to have college degrees, or some college education. We also see the geographical distribution for men and women is similar. We next proceed to answer our question.

We estimate the linear regression model that we have introduced. Our outcome Y is hourly wage, and our target regressor D is the indicator of being a female. And our control's W's are the job-relevant characteristics. We consider two specifications for the controls. In the basic model, our W's consist of education and regional indicators, experience, experience squared, and experience cubed.

In the flexible model, our W's consist of controls in the basic models and all of their two-way interactions. We now go to the empirical results. In the following table, we see the estimated regression coefficient, its standard error and the 95% confidence interval for both basic and flexible regression models.

The results for the two regression models is very close degree. The estimated gender gap in hourly wage is about -\$2. This means that women get paid \$2 less per hour on average than men, controlling for experience, education, and geographical region. The 95% confidence interval arranges for -2.7 to minus \$1.

We conclude that he difference in the hourly wage for men and women who have the same recorded characteristics is both statistically and economically significant. To sum it up, we applied the ideas we discussed so far to learn about the gender wage gap. The gender wage gap may partly reflect gender discrimination against women in the labor market.

It may also partly reflect the so called selection effect, namely that women are more likely to end up in occupations that pay somewhat less. For example, in teaching. This segment concludes our discussion of the classical linear regression. Next, we will briefly discuss other types of regression.

2.1.7 Other Types of Regression

In this segment, we'll briefly discuss other types of regression. First, we will introduce regressions introduced by using nonlinear functional forms, for example logistic regression. Second, we will mention regressions that result from replacing the squared loss function by other loss functions. For example, using the absolute deviation loss gives rise to median regression, and using the estimatric absolute deviation laws gives rise to quantile regression.

In contrast to linear regression, in nonlinear regression we use a nonlinear function of parameters and regressors, say P of X and beta to predict Y. For instance, the logistic regression employs the logistic transformation of beta X given by the formula that you see. This logistic transformation is particularly attractive when the response variable, Y, is binary taking on values of zero or one so that the predicted values are naturally constrained between zero and one.

The problem of predicting binary Y is a basic example of a classification problem. We can interpret the predicted values in this case as approximating probability that Y=1. In nonlinear aggression, we estimate the parameters by solving the sample nonlinear least squares problem. Where we minimized the average squared error for predicting Yi by p(Xi, b).

In the case of binary outcomes, we often estimate the parameters by maximizing the logistic log-likelihood function for the binary outcomes Yi conditional on Xi as you see in this formula. We previously used the squared error loss to set up the best linear predictor. Sometimes we call this approach the linear mean regression, because the best predictor of Y under squared loss is the conditional mean of Y given X.

So what if we use the absolute deviation loss instead? Then we obtain the least absolute deviation regression or median regression, because the best predictor of Y under absolute deviation loss is the median value of Y conditional on X. We define the linear median regression by solving the best linear prediction problem under the absolute deviation loss.

Where in the population we use the theoretical expectation and in the sample we use the empirical expectation instead. Just like mean regression, median regression can also be made nonlinear by replacing beta X with some nonlinear function p (X, b). Regarding the motivation, median regressions are especially great for cases when the outcomes have heavy tails or outliers.

The median regressions have much better properties in this case than the mean regressions. If we use an asymmetric absolute deviation loss, then we end up with the asymmetric absolute deviation regression, or quantile regression. Specifically we define the linear Tao quantile regression by solving the best linear prediction problem under the asymmetric absolute deviation loss.

In the population we use the population expectation and in the sample we use the empirical expectation instead. The weight Tao that appears in the definition of the absolute deviation loss specifies that Tao times 100-th percentile of Y given that we are trying to model linearly. Quantile regressions are great for determining the factors that affect the outcomes in the tails.

For example, in risk management, we might predict the extremal conditional percentiles of Y using the information X. This type of prediction is called the conditional value-at-risk analysis. In medicine, we could be interested in how smoking and other controllable factors X affect very low percentiles of infant birth weights Y.

In supply chain management, we could try to predict the inventory level for a product that is able to meet the 90-th percentile of demand Y given the economic conditions described by X. In this segment, we have briefly overviewed non-linear regressions as well as quantile regressions and their uses.

In the next block of our module, we will consider modern linear and nonlinear regressions which are motivated by high-dimensional data.

2.2 Modern Regression with High-Dimensional Data

2.2.1 Modern Linear Regression for High-Dimensional Data

We begin a new block of segments where we consider more than linear regression for high-dimensional data. We consider the linear regression model Y = beta X + epsilon where the beta X is the population best linear predictor, or BLP, of Y using X, or simply the population linear regression function.

An epsilon is uncorrelated with X. Here, the regressor X is p-dimensional. That is, there are p regressors, and the sample size is n, and p is large, possibly larger than n. One reason for having high-dimensional regressors in our model is the increasing availability of modern rich data or, quote, unquote, big data.

Many more than data sets are rich that they have many recorded features or regressors. For example, in house pricing and appraisal analysis, we can make use of numerous housing characteristics. In demand analysis, we can rely on price and product characteristics. And in the analysis of court decisions, we can employ many of the judge's characteristics.

Another reason for having high-dimensional regressors in our model is the use of constructed regressors. By this we mean that is zet are raw regressors or features then the constructed regressors X are given by the set of transformations P(zet) whose components are PJ(zet). We sometimes call this set of transformations a dictionary.

For instance, in the wage example, we use quadratic and cubic transformations of experience, as well as interactions or products of these regressors with education and geographic indicators. The use of constructed regressors allows us to build more flexible and potentially better prediction rules than just linear rules that employ raw regressors.

This is because we're using prediction rules, beta X = beta P(zet), that are allowed to be either linear or nonlinear in the raw regressor zet. We'll now note that we still call the prediction rule beta X linear because it is linear with respect to the parameters beta and with respect to the constructed regressors X = P(zet).

We'll next ask the following related question. What is the best prediction rule for Y using zet? It turns out that the best prediction rule is the conditional expectation of Y given zet. We denote this prediction rule by g(zet), and we call it the regression function of Y on zet.

This is the best prediction rule among all rules, and it is generally better than the best linear prediction rule. Indeed, it can be shown that g(zet) solves the best prediction problem, where we minimize the mean squared prediction error, or MSE, among all prediction rules m(zet), (linear or nonlinear in zet).

Now, by using beta P(zet), we are implicitly approximating the best predictor g(zet). Indeed, it can be shown that for any parameter value b, the MSE for predicting Y- b zet = the MSE for approximating g(zet)- b P(zet) + a constant. That is, the mean squared prediction error is equal to the mean squared approximation error, plus a constant that doesn't depend on b so that minimizing the former is the same as minimizing the latter.

We now conclude that the BLP beta P(zet) is the best linear approximation, or BLA, to the regression function g(zet). By using richer and richer dictionary of transformations P(zet), we can make the BLP beta P(zet) approximate g(zet) better and better. We can illustrate this point by the following simulation sample.

Suppose zet is uniformly distributed on the unit and the true regression function g(zet) is the exponential function(4.zet). Suppose we don't know this, and we use the linear forms beta P(zet) to provide approximations to g(zet). Suppose we use P(zet) that consists of polynomial transformations of zet consisting of the first P terms of the polynomial series, 1, zet, zet squared, zet cubed, and so on.

We use this dictionary to form the BLA or BLP beta P(zet). We now provide a sequence of figures to illustrate the approximation properties of the BLA or BLP corresponding to P(zet) = 1, 2, 3, and 4. With p = 1, we'll get a constant approximation to the regression function g(zet).

And as we can see, the quality of this approximation is very poor. With p = 2, we get a linear in zet approximation to g(zet). And as the figure shows, the quality of this approximation is still very poor. With p = 3, we get a quadratic in zet approximation to g(zet).

And now the quality is quite good all of a sudden. This explains why using nonlinear transformations over raw regressors is a good idea. Now with p = 4, we get a cubic in zet approximation to g(zet), and the quality of approximation becomes simply epsilon. This further stresses the motivation for using nonlinear transformations of raw regressors in linear regression analysis.

In summary, we provided two motivations for using high-dimensional regressors in prediction. The first motivation is that modern data sets have high-dimensional features that can be used as regressors. The second motivation is that we can use nonlinear transformations of features or raw regressors and their interactions to form constructed regressors.

This allows us to better approximate the ultimate and best prediction rule, the conditional expectation of the outcome given raw regressors.

2.2.2 High-Dimensional Sparse Models and Lasso

In this segment, we will talk about high-dimensional sparse models and a penalized regression method called the Lasso. Here, we consider the regression model Y = beta X + epsilon, where beta X is a population-best linear predictor of Y using X. Or simply the population linear regression function.

The regressor X is p-dimensional, with components denoted by Xj. That is, there are p regressors, and p is large, possibly much larger than N, where N is the sample size. In order to simplify the discussion, we assume that each Xj has a unit variance in the sample. Here, we are dealing with the high-dimensional setting, where p/N is not small.

Classical linear regression, or ordinary least squares fails in these settings, because it overfits the data, as we have seen in part one of our module. We need to make some assumptions and modify the classical regression method to deal with the high-dimensional case. One intuitive assumption is approximate sparsity, which informally means that there is a small group of regressors that have relatively large coefficients, that can be used to approximate the BLP beta 'X quite well.

The rest of regressors have relatively small coefficients. An example of an approximately sparse model is the linear regression model with the regression coefficients beta j given by 1/j-squared as j ranges from 1 to p. The figure that you see here shows the graph of these coefficients. As we can see, the coefficients has decreased quite fast, with only three or four regression coefficients that appear to be large.

Formally, the approximate sparsity means that the sorted absolute values of the coefficients decrease to zero fast enough. Namely, the js largest in absolute value coefficient is at most of size j into the power of minus a times a constant, where a is greater than one half. Here, the constant a measures the speed of decay.

For estimation purposes, we have a random sample of Yi and Xi, where i ranges from 1 to n. We seek to construct a good linear predictor, head-beta X, which works well when p over N is not small. We can construct head-beta as a solution of the following penalized regression problem called Lasso.

Here, we are minimizing the sample mean-squared error that results from predicting Yi with bX plus a penalty term, which penalizes the size of the coefficients, bjs, by their absolute values. We control the degree of penalization by the penalty level lambda. A theoretically justified penalty level for Lasso is given by the formula that you see here.

This penalty level ensures that the Lasso predictor, hat beta X, does not overfit the data and delivers good predictive performance under approximate sparsity. Another good way to pick penalty level is by cross-validation, which uses repeated splitting of data into training and testing samples to measure predictive performance. We will discuss cross-validation later in this module.

Intuitively, Lasso imposes the approximate sparsity on the coefficients hat beta, just like in the assumption. It presses down all of the coefficients to zero, as much as possible without sacrificing too much fit, and it ends up setting many of these coefficients exactly to zero. We can see this in the simulation example where our Xjs and epsilons are standard normal variables, and the true coefficients beta j are equal to 1 over j squared.

Suppose also that n is equal to 300 and p is equal to 1000. The following figure shows that hat beta, in blue, is indeed sparse and is close to the true coefficient vector beta, in black, indeed. Most of hat-beta js are set to 0, except for several coefficients that align quite well with the largest coefficient beta j, or the true coefficient vector.

This really shows that Lasso is able to leverage approximate sparsity to deliver good approximation to the true coefficients. From the figure, we see that Lasso sets most of the regression coefficients to zero. It basically figures out approximately, though not perfectly, the right set of regressors. In practice, we often use the Lasso-selected set of regressors to refit the model by least squares.

This method is called the least squares post Lasso, or simply post-Lasso. Post Lasso does not shrink large coefficients to zero as much as Lasso does. And it often improves over Lasso in terms of prediction. We next discuss the quality of prediction that Lasso and Post-Lasso methods provide. In what follows, we will use the term Lasso to refer to either of these methods.

By definition, the best linear prediction rule, out-of-sample, is beta X, so the question is, does hat-beta X provide a good approximation to beta X? We are trying to estimate p parameters, beta 1 through beta p, imposing the approximate sparsity via penalization. Under sparsity, only a few, say s, parameters will be important.

And we can interpret s as the effective dimension. Lasso approximately figures out which parameters are important, and estimates them. Intuitively, to estimate each of the important parameters well, we need many observations per each size parameter. This means that n over s must be large, or equivalently, s over n must be small.

This intuition is indeed supported by the following theoretical result, which reads, under regularity conditions, the root of the expected square difference between the best linear predictor and the Lasso predictor is bounded about by a constant times the level of noise, times square root of the effective dimension s times lower p n divided by n.

Here, we are averaging over the values of x and the bond holds probability close to one for large enough sample sizes. The effective dimension s is equal to constant times n into the power of 1 over 2a, where a is the rate of decrease of coefficients in the approximate sparsity assumption.

In other words, if n is large and the effective dimension s is much smaller than n over log(pn), for nearly all realizations of the sample, the Lasso predictor gets really close to the best linear predictor. Therefore, under approximate sparsity, Lasso and Post-Lasso will approximate the best linear predictor well.

This means that Lasso and Post-Lasso won't overfit the data, and we can use the sample and adjusted R squared and MSE to assess out-of-sample predictive performance. Of course, it's always a good idea to verify out-of-sample predictive performance by using test or validation samples. So, let us summarize. We have discussed approximate sparsity as one assumption that makes it possible to perform estimation and prediction with high-dimensional data.

We have introduced Lasso, which is a regression method that imposes approximate sparsity by penalization. Under approximate sparsity, Lasso is able to approximate the best linear predictor, and thus produces high quality prediction.

2.2.3 Inference with Lasso

In this segment we discuss how to use loss to answer the inference question. Which is how the predictive value of y change if we increase the component of t(x) by holding the other components of x fixed. The answer is a population regression coefficient Beta 1 corresponding to the regressor D.

And here would like to discuss how to use Lasso to estimate and construct confidence intervals for beta 1 in the high dimensional setting. We write the regression equation as Y equals beta 1D plus beta 2 w plus epsilon, where d is the target regressor and w's our d controls.

We recall from part one of our module that after portioning out we ended up with a simplified regression equation. Tilda Y = tilda d beta 1 + epsilon where epsilon is uncorrelated with tilda d. Here, tilde Y and tilde D, are the residuals that are left after parceling out the linear effect of W.

As we argued in part one, this allows us to obtain beta 1 as a coefficient in the linear regression of tilde Y and tilde D. D in the population recovers the partialling-out procedure. For estimation purposes, we have a random sample of Yis and Xis. Our main idea here is that, we will mimic in the sample the partialling-out procedure in the population.

Previously, when p over n was small, we employed least squares as the Prediction method in the partialling-out steps. Here p over n is not small, and we employed instead the Lasso method in the partialling-out steps. So let us explain this in more detail. First we round the lasso regression of Yi on Wi and of Di on Wi and keep the resulting residuals called check Wi and check Di.

Second we run the list squares of check Wi on check Di. The resulting estimated regression confusion is our estimator check beta one. Our portioning out procedure was loss all. It relies on approximate sparcity in the two portioning out steps. Indeed, theoretically, the procedure will work well if the population regression coefficients and the two parceling out steps are approximately sparse, with a sufficiently high speed of degrees of assorted values, as shown in the conditions that you see here.

Now we present the following theoretical result which reads, Under the stated approximate sparsity and other regularity conditions, the estimation error in check Di and check Yi has a negligible effect on check beta 1. And check beta 1 is approximately distributed as normal variable with mean beta 1 and variance V over n where expression of the variance appears here.

That is we can say that the estimator check beta 1 concentrates in a square root of V / n neighborhood of beta 1 with deviations controlled by the normal law. We can use this result just like we used the analogous result for the low dimensional case in block one.

We can define the standard error of check beta one as square root of hat v over n. Where hat v is an estimator of v. This is our quantification of uncertainty about that one. We then provide the approximate 95% confidence interval for beta 1, which is given by the estimate check beta 1 + or- two standard errors.

So let's give a summary. In this segment, we learnt how to estimate the target regression estimate, beta one, in the high dimensional regression problem. We use Lasso to obtain estimates of the outcome Y and target regression D net of the effect of the effect of other regressors W and then run least squares of one on the other.

Then we argued that the resulting estimator check beta1 is high quality estimator beta1 and we constructed a confidence interval for beta1,

2.2.4 Case Study: Do Poor Countries Grow Faster Than Rich Countries?

In this segment, we provide an empirical example of using partialling-out with Lasso to estimate the regression coefficient beta 1 in the high-dimensional linear regression model, Y = beta 1 D + beta 2 W + epsilon. Specifically, we are interested in how the rates in which economies of different countries grow, denoted by Y, are related to the initial wealth levels in each country, denoted by D.

Controlling for countries institutional, educational, and other similar characteristics, denoted by W. The relationship is captured by the regression coefficient beta one. In this example, this coefficient is called the speed of convergence or divergence, as it measures the speed at which poor countries catch up or fall behind wealthy countries, controlling for W.

So our inference question here is do poor countries grow faster than rich countries, controlling for educational and institutional characteristics. In other words, is this bit of convergence negative, namely, is beta one negative? In economics, this question is known as the Convergence Hypothesis, which is predicted by the Solow Growth Model.

The growth model was developed by Professor Robert M Solow, a world renowned MIT economist, who won the Nobel Prize in Economics in 1987. In this case study, we use the data collected by economist Robert Barro and Jong-Wha Lee. In this data set, the outcome y is the realized annual growth rate of a country's wealth measured by the gross domestic product per capita.

The target regressor D is the initial level of the country's wealth, the controls W include measure of the education levels, quality of institutions, trade openness, and political stability in the country. The data sets contains 90 countries and about 60 controls. So P is approximately 60 and N is 90, and P over N is not small.

This means that we operate in a high dimensional setting, therefore, we expect in this squares method to provide a poor, very noisy estimate of beta one. And in contrast, we expect the method based on partialling-out was Lasso to provide a high quality estimate of beta one. We now present the empirical results in the table that you can see here.

The table shows the estimates of the speed of conversions beta one, obtain value squares and by partialling-out was lasso. The table also provides the standard errors and 95% confidence intervals. As we expected, the least squares method provides a rather noisy estimate of the annual speed of convergence and does not allow us to answer the question about the convergence hypothesis.

In sharp contrast, partialling out via lasso provides a much more precise estimate and does support the convergence hypothesis. We see that the lasso-based estimate of error 1 Is minus 4% and the 95% confidence interval for the annual rate of convergence is from minus 7.5% to minus 1.5%. Now this empirical evidence does support the convergence hypothesis.

Let us summarize, in this segment, we have examined an empirical example in the high-dimensional setting. Using least squares in the setting gives us a very noise estimate of the target regression coefficient, and does not allow us to answer an important empirical question. In sharp contrast, using the partialling out method with Lasso does give us a precise estimate of the regression coefficient, and does allow us to answer the question.

We have found significant empirical evidence supporting the convergence hypothesis of Solow.

2.2.5 Other Penalized Regression Methods. Cross-Validation

In this segment, we overview other penalized regression methods and also discuss cross-validation, which is a method to choose tuning parameters for the prediction rules. We are interested in prediction in the linear model Y = beta X + epsilon, where epsilon isn't correlated with X. And we have random sample of Yi and Xis.

Our generic predictor will take the linear form hat f of x = hat beta x. The idea of penalized regression is to choose the coefficients hat beta, to avoid overfitting in the sample. This is achieved by penalizing the size of the coefficients by various penalty functions. Ideally, we should choose the level of penalization to minimize the out-of-sample mean squared prediction error.

We first consider the Ridge method. The Ridge method estimates coefficients by penalized least squares, where we minimize the sum of squared prediction error plus the penalty term given by the sum of the squared values of the coefficients times a penalty level, lambda. We can see this in the formula given here.

If we look at the formula, we notice that analogous to the Lasso, Ridge penalty presses down or penalizes the coefficients without sacrificing too much fit. In contrast to Lasso, Ridge penalizes the large values of coefficients much more aggressively and small values much less aggressively. Because Ridge penalizes small coefficients very lightly, the Ridge fit is never sparse.

And unlike Lasso, Ridge does not set estimated coefficients to zero, and so it does not do variable selection. Because of this property, Ridge predictor hat beta X is especially well suited for prediction in the dense models, where the beta js are all small without necessarily being approximately sparse.

In this case, it can easily outperform the Lasso predictor. Finally we noted in practice, we can choose the penalty level lambda in Ridge, by cross-validation which we will discuss later in this segment. A Ridge and Lasso have other useful modifications or hybrids. One popular modification is the method called the elastic net.

Here we estimate the coefficients by penalized least squares with penalty given by the linear combination of the Lasso and the Ridge penalties as you see in this formula. In the formula, we see that the penalty function has two penalty levels, lambda 1 and lambda 2, which could be chosen by cross-validation in practice.

Now, let us look at the formula carefully. We see that the elastic net penalizes large coefficients as aggressively as Ridge. And we also see that it penalizes small coefficients as aggressively as

Lasso. By selecting different values of penalty levels, lambda 1 and lambda 2, we could have more flexibility for building a good prediction rule than with just Ridge or with just Lasso.

We also note that the elastic net doesn't perform variable selection, unless we completely shutdown the Lasso penalty by setting penalty level lambda 2 equals zero. Elastic net works well in regression models, where regression coefficients are either approximately sparse or dense. Another useful modification of Lasso and Ridge is the lava method.

In the lava method, we estimate the coefficients by the penalized least squares as shown in this form. If we look at the formula carefully, we see that, just like previously, we are minimizing the sum of squared prediction errors from predicting outcome observations Yi with a linear and Xi prediction rule plus a penalty term.

However, unlike previously, we are splitting the parameter components into gamma j, + delta j, and penalize gamma j like in Ridge and delta j like in Lasso. There are two corresponding penalty levels, lambda 1 and lambda 2, which can be chosen by cross-validation in practice. Now, because of the splitting, lava penalizes coefficients least aggressively compared to Ridge, Lasso, or elastic net, because it penalizes large coefficients like in Lasso and small coefficients like in Ridge.

Lava never sets estimated coefficients to zero, and so it doesn't do variable selection. The lava method works really well in sparse + dense models, where there are several large coefficients and many small coefficients, which are not necessarily sparse. In these types of models, lava significantly outperforms Lasso, Ridge or elastic net.

We have all ready mentioned cross-validation several times during the course of the segment. Cross-validation is an important and common practical tool that provides a way to choose tuning parameters such as the penalty levels. The idea of cross-validation is to rely on the repeated splitting of the training data to estimate the potential out-of-sample predictive performance.

Cross-validation proceeds in three steps, which we've first described in words as follows. In step 1, we divide the data into K blocks called folds. For example was, K equal to five, we split the data into five parts. In step 2, we begin by leaving one block out. We fit the prediction rule on all other blocks, we then predict outcome observations in the left out block and record the empirical mean squared prediction error.

We repeat this procedure for each block. In step 3, we average the empirical mean squared prediction errors over blocks. We do these steps for several or many values of the tuning parameters, and we choose the best tuning parameter to minimize the average mean squared prediction error. Let us now consider more formal description of cross-validation.

We randomly select equal sized blocks B1 through Bk to randomly partition the observation indices one through M. We then fit a prediction rule denoted by hat f sub -k or X and theta. Where theta denotes tuning parameters such as penalty levels, and hat f sub -k depends only on observations that are not in the block Bk.

The empirical mean squared error for the block of observations Bk is given by the average squared prediction error for this block, as shown in this formula. In this formula M is the size of the block. The cross validated MSE is the average of MSEs for each block as shown again, in this formula.

Finally, the best tuning parameter theta is chosen by minimizing the cross validated MSE. We now provide some concluding remarks. First, we note that in contrast to Lasso, the theoretical properties of Ridge and other penalized procedures are less well understood in the high-dimensional case, yet. So it is a good idea to rely on test data to assess their predictive performance.

Second, we note that cross validation is a good way to choose penalty levels, but its theoretical properties are not completely understood in high-dimensional case yet. So again, it is a good idea to rely on task data to assess the predictive performance of cross-validated rules. Finally, we may want to ask a question here.

How do the penalize regression methods work in practice? In the next part of our module we will assess the predictive performance of these methods in a real example, where we will also compare these methods to modern nonlinear regression methods.

2.3 The Use of Modern Regression for Causal Inference

2.3.1 Modern Nonlinear Regression. Trees, Random Forests, and Boosted Trees

We begin a new block of our module devoted modern nonlinear regression. Here we are interested in predicting the outcome y using the raw regressor Zed which are k-dimensional. Recall that the best prediction rule g(Z) is the conditional expectation of y given Z. In this module so far, we have used best linear prediction rules to approximate g(Z) and linear regression or lost regression for estimation.

Now we consider nonlinear prediction rules to approximate g(Z), including tree-based methods and neural networks. In this segment we discussed tree-based methods. The idea of regression trees is to partition the regressor space where Z takes values into a set of rectangles, and then for each rectangle provide a predicted value.

Suppose we have n observations, Zi, Yi for i ranging from 1 to n. Given the partition of the space into M rectangles, R1 through Rm, which will be determined by data. The regression rule is a prediction of the four. Hat g(Z) is equal to the sum over m from 1 to m of hat beta m times the indicator that Z belongs to the rectangle Rm.

The estimated coefficients are obtained by minimizing the sample MSE, as shown in this formula. From the formula we can conclude that hat Beta m is set equal to average of Yi with Zi falling in the rectangle Rm, as shown in the formula that you see. The rectangles or regions Rm are called nodes and each nodes has predicted value had Rm associated with it.

And nice thing about regression trees is that you get to draw cool pictures like this one. Here we show a tree based prediction rule for our wage example where Y is wage and Z are experience, geographic, and educational characteristics. As you can see from looking at the terminal nodes of the tree, in this tree, the predicted hourly wage for college graduates with more than 9.5 years of experience is $24, and otherwise it is 17.

The predicted wage for non-college graduates with more than 14 years of experience is 14, and otherwise it is 12. Now how do we grow this tree? To make computation tractable, we use the recursive binary partitioning or splitting of the regressor space. First, we cut that regressor space into two regions by choosing a regressor and a split point that achieve the best improvement in the sample MSE.

This gives us the tree of depth one, that you see in the figure. The best variable to split on here is the indicator of college degree and it takes values of 0 or 1, so the natural split point is 0.5. This gives us a starting tree-based prediction rule, which predicts a $20 hourly wage for college graduates and 13 for all others.

Second we repeat this procedure over two resulting regions or nodes, college graduates and non-college graduates, obtaining in this step four new nodes that we see in this figure. For college graduates the splitting rule that minimizes MSE is the experience regressor at 9.5 years. This refines the prediction rule for graduates to $24 if experience is greater than 9.5 years and $12 otherwise.

For non-graduates the procedure works similarly. Third, we repeat this procedure over each of the four nodes. The tree of that three now has eight nodes. We see that in the final level we are now splitting our gender indicator, high school graduate indicator, and the south indicator. Finally, we stop growing the tree when the desired depths of the tree is reached.

Or when the minimal number of observations per node, called minimal node, size is reached. We've now made several observations. First, the deeper we grow the tree, the better is our approximation to the regression function g(Z). On the other hand, the deeper the tree, the more noisy our estimate of g(Z) becomes, since there are fewer observations per terminal node to estimate the predicted value for this node.

From a prediction point of view, we can try to find the right depth or the right structure of the tree by cross-validation. For example, in the wage example the tree of depth 2 performs better in terms of cross-validated MSE than the trees of depths 3 or 1. The process of cutting down the branches of the tree to improve predictive performance is called Pruning The Tree.

However in practice pruning the tree often doesn't give satisfactory performance because a single prune tree provides a very crude approximation to the regression function g(Z). A much more powerful and one use approach to improve simple regression trees is to build the Random Forest. The idea of Random Forest is to grow many different deep trees and then average prediction rules based on them.

The trees are grown over different artificial data samples generated by sampling randomly with replacement from the original data. This way of creating artificial samples is called the bootstrap statistics. The trees are growing deep to keep the approximation error low and averaging is meant to reduce the noisiness of the individual trees.

Let us define the bootstrap method. Each bootstrap sample is created by sampling from our data on pairs (Yi, Zi) randomly with replacement, so some observations get drawn multiple times and some don't get redrawn at all. Given a bootstrap sample numbered by numbered by numeral b, we build a tree-based prediction rule hat gb (Z).

We repeat the procedure capital B times in total and then average the prediction rules, that result from each of the bootstrap samples. So here we have hat g random forest (Z) equals 1 over B times the sum of hat g sub B of Z over B, where b runs from one to capital B.

Using bootstrap here is an intuitive idea. If we could have many independent copies of the data, we could average the prediction rules obtained over these copies to reduce the noisiness. Since we don't have such copies, we rely on bootstrap copies of the data. The key underlying idea here is that the trees are grown deep to keep the approximation error low and averaging is meant to reduce the noisiness of the individual trees.

The procedure of averaging noisy prediction rules over the bootstrap samples is called Bootstrap Aggregation or Bagging. Finally I would like to know that what we discuss here is the simplest version of the random forest, and there are many different modifications aimed at improving predictive performance that we didn't discuss.

But I'm encouraging you to look at the course materials for additional references that discuss random force and more detail. Another approach to improve simple regression tree is by boosting. The idea of boosting is that of request of fatigue where estimated tree-based prediction rule, then we take the residuals and estimate another shallow tree-based prediction rule for these residuals and so on.

Summing up the prediction rules for the residuals gives us the prediction rule for the outcome. Unlike in a random forest, we use shallow trees, rather than deep trees, to keep the noise low and each step of boosting aims to reduce the approximation error. In order to avoid overfitting and boosting, we can stop the procedure once we don't improve the cross-validated mean square error.

Formerly, the boosting algorithm looks as follows. In step 1, we initialize the residuals Ri = Yi, for i that runs from 1 to n. In step 2, we perform the following operation over index j running from 1 to capital J. We fit a tree-based prediction rule, gj(Z) to the data (Zi,Ri) with i from 1 to n and we have data residuals as R1 equals previous Ri minus lambda times hat g sub j (Zi).

Finally in step 3, we output the boosted prediction rule. Hat g(Z) equals sum over J of lambda hat g sub j(Z). The capital J and lambda that you see here are the tuning parameters. Representing the number of boosting steps, and the degree of updating the residuals. In particular, lambda equals 1, gives us the full update.

We can choose j and lambda by cross validation. So let us summarize. We discussed the tree based prediction rules, and ways to improve them by bagging or boosting. Bagging is bootstrap aggregation of the prediction rules. Bootstrap aggregation of deep regression trees gives us random forests. Boosting uses recursive fitting of residuals by a sequence of tree-based prediction models.

The sum of these prediction rules gives us the prediction for outcome.

2.3.2 Modern Nonlinear Regression. Neural Networks

In this segment, we consider a non linear regression method, based on neural networks. As before, we are interested in predicting the outcome Y using the raw regressors Z, what are K-dimensional. Recall that the best prediction rule of Y given Z is the function g(Z), the conditional expectation of Y given Z.

The idea of the neural network is to use parameterized nonlinear transformations of linear combinations of the raw regressors as constructed regressors, called neurons. And produce the predicted value as a linear function of these regressors. The prediction rule is g(Z) is nonlinear in some parameters and with respect to raw regressors.

With sufficient many neurons neurons, g(Z) can approximate the best prediction rule g(Z). In part 2 of our module, we already saw that many constructed regressors are useful in the high-dimensional linear setting to approximate g(Z). Neural networks also rely on many constructed regressors to approximate g(Z). The method and the name neural networks were loosely inspired by the mode of operation of the human brain, and developed by the scientists working on Artificial Intelligence.

Accordingly, neural networks can be represented by cool graphs and diagrams that we will discuss shortly, so please stay tuned. To discuss the idea in more detail, we focus on the single layer neural network. The estimated prediction rule will take the form hat g(Z) equals sum over M, running from 1 to capital M of hat betaM times Xm of hat alphaM.

Where the Xm of hat alpha m's are constructed regressors called neurons. The capital M neurons in total. The neurons are generated by the formula Xm of alpha m = sigma of alpha mZ. Where alpha m's are neuron-specific vectors of parameters called weights and sigma is the activation function.

For example, sigma can be the sigmoid function given by this formula. Or it can be the so called rectified linear unit function or ReLU given by this formula. The following figure shows the two graphs of the two Activation Functions. The sigmoid function and the rectified linear unit function.

The horizontal axis shows the value of the argument. And the vertical axis the value of the function. The estimators had alpha m, and beta m for each M are obtained as the solution to the penalized nonlinear least squares problem shown by this formula. Here, we are minimizing the sum of squared prediction erros in the sample, plus a penalty term given by the sum of the absolute values of components of alpha m and beta m, multiplied by the penalty level, lambda.

In this formula, we use the lasso type penalty but we can also use the ridge, another type of penalties. The estimates are computed using sophisticated gradient descent algorithms. Where sophistication is needed because nonlinear least squares optimization problem is generally not a convex problem. Making the computation a difficult task.

The procedural fetching renewal network model has tuning parameters and in practice we can choose them by cross-validation. The most important choices concern the number of neurons and the number of neuron layers. Having more neurons gives us more flexibility, just like having more constructed regressors gave us more flexibility with high-dimensional linear models.

To prevent overfitting, we can rely on penalization as in the case of linear regression. In order to visualize working of the neural network, we rely on the resource called Playground.TensorFlow.org using which we produce a prediction model given by simple single layer neural network model. We now see the graphical representation of this network.

Here we have a regression problem and the network depicts the process of taking row regressors and transforming them into predicted values. In the second column on the left, we see the inputs are features. These features are our two row regressors. The third column shows eight neurons. The neurons are constructed as linear combinations of the row regressors transformed by an activation function.

That is, the neurons are along linear transformations of the row regressors. Here, we set the activation function, to be the rectified linear unit function, RE of U. The neurons are connected to the inputs and the connections represent the coefficients hat alpha M, which are coefficients of the neuron specific linear transformations of raw regressors.

The coloring represents the sign or the coefficients, orange negative and the blue positive. And with all the connections represents the size of the coefficients. The neurons are then linearly combined to produce the output, the prediction rule. In the diagram we see the connections going outwards from the neurons to the output.

These connections represent the coefficients hat beta M, or the linear combination of the neurons that produce the final output. The coloring and the widths represent the sign and the size of these coefficients. In the diagram, the prediction rule is shown by the heatmap in the box on the right.

On the horizontal and vertical axis, we see the values of the two inputs. The color and its intensity in the heatmap represent the predicted value. We also see on the top that the penalty function is L1, which stands for the lasso type penalty. Another option is to use L2, which stands for the rich type penalty.

The penalty level is called here the regularization rate. In this example we are using a single layer neural network. If we add one or two additional layers of neurons constructed from the previous layer of neurons, we get a different network, which we show in the following diagram. Prediction methods based on neural networks with several layers of neurons called the deep learning methods.

This diagrams showcases that one of the major benefits of doing prediction with neural networks is that you can adapt with pretty cool artwork. Let us summarize, in this segment we have discussed neural networks that have recently gathered much attention under the umbrella of deep learning. Neural networks represent a powerful and all-purpose method for prediction and regression analysis.

Using many neurons and multiple layers gives rise to networks that are very flexible and can approximate the best prediction rules quiet well.

2.3.3 Aggregation of Predictors. Case Study: Predicting Wages 2

In this segment, we'll discuss the quality of prediction that the modern methods provide. We begin by recalling that the best prediction rule, for outcome Y, using features or regressor Z, is the function g of Z equal to the conditional expectation of Y using Z. Modern regression methods, namely Lasso, Random Forest, Boosted Trees, Neural Networks, when appropriately tuned and under some regularity conditions, provide estimated prediction rules hat g (Z) that approximate the best prediction rule g(Z).

Theoretical work demonstrates that under appropriate regularity conditions and with appropriate choices of tuning parameters, the mean squared approximation error can be small once the sample size n is sufficiently large. These results do rely on various structured assumptions, such as sparsity in the case of Lasso, and others to deliver these guarantees in modern, high dimensional settings where the number of features is large.

From a practical standpoint, we expect that under these conditions, the sample MSE and R squared will tend to agree with the out-of-sample MSE and R squared. We can measure the out-of-sample performance empirically by performing data splitting, as we did in the classical linear regression. Recall that we use a random part of data, say half of data, for estimating or training the prediction rules.

We use the other part of data to evaluate the predictive performance of the rule, recording the out-of-sample MSE, or R-squared. Accordingly, we call the first part of data the training sample, and the second part of data the testing or validation sample. Indeed, suppose use n observations for training and m for testing or validation.

And let capital V denote the indices of observations in the test sample. Then the out-of-sample or test mean squared error is defined as the average squared prediction error, what will predict Yk in the test sample by hat g(Zk), where the prediction rule hat g was computed on the training sample.

The out-of-sample R square is defined accordingly as 1 minus the ratio of the test MSE to the variation of the outcome in the test sample. We next illustrate the ideas using a dataset of 12,697 observation from the March Current Population Survey Supplement 2015. In this data set, the outcome observations, Yi's, are log wages of never-married workers living in the United States, and the feature Zi's consist of a variety of worker characteristics, including experience, race, education, 23 industry and 22 occupation indicators, and some other characteristics.

We will estimate or train two sets of prediction rules, linear and non-linear models. In linear models, we estimate the prediction rule of the form hat g(Z) equals hat beta prime X, with X generated in two ways. In the basic model, X consists of 72 raw regressors Z and a constant.

In the flexible model, X consists of 2336 constructive regressors, which includes the lower regressor Z, four polynomials in experience and all two-way interactions of these regressors. We estimate hat beta by linear regression or least squares and by penalized regression methods. Namely Lasso, Post-Lasso, Cross-Validated Lasso, Ridge, and Elastic Nets.

In nonlinear models, we estimate the prediction rule of the form hat g(Z). We estimate them by Random Forest, Regression Trees, Boosted Trees, and Neural Network. We present the results in the table that you see. We report the results for a single equal spread of data to the training and testing part.

The table shows the test MSE in column one as well as the standard error for the MSE in column two, and the test R-squared in column three. We see that the prediction rule produced by Random Forest performs the best here, giving the lowest out-of-sample, or test MSE and the highest test R squared.

Other methods, for example, Lasso, Cross-Validated Elastic Net, Boosted Trees, perform nearly as well. They all perform similarly with the test MSEs being within one standard error of each other. Remarkably the simple classical OLS on a simple model with 72 regressors performs extremely well, almost as well as Random Forest.

Since the performance of OLS done on a simple model is statistically indistinguishable from the performance of the Random Forest, we may choose this method to be the winner here. Other the hand, classical OLS done on a flexible model with 2335 regressors performs very poorly. It provides a prediction rule that is very noisy and imprecise.

Penalized regression methods on the flexible model such as VASA, do much better because they are able to reduce the imprecision, while leveraging the low approximation error of the flexible model. Pruned regression tress and simple neural networks (with a small number of neurons) don't perform well here. This is because these methods provide an approximation to the best prediction rule that is too crude, resulting in too much bias relative to other methods.

Given the results presented, we can choose one of the best performing prediction rules. For example, we can select the prediction rule generated by least squares on the simple model, or the prediction rule generated by Lasso on the flexible model, or the prediction rule generated by the random forest.

We can also consider aggregations or ensembles of prediction rules, which combine several prediction rules into one. Specifically, we can consider the aggregated prediction rule of the form tilde-g(Z) = sum over k running from 1 to K of tilde alpha k hat gk(Z). Where hat gk is our prediction rules obtained using the training data, including possibly a constant, and tilde alpha ks are the coefficient assigned to the different prediction rules.

We can build K prediction rules from the training data. We can figure out the aggregation coefficients, tilde alpha ks, using the test validation data. If the number of prediction rules, K, is small, we can figure out the aggregation coefficients using test data by simply running least squares of outcomes on predicted values in the test sample.

Where we minimize the sum of squared prediction errors on predicting Yi by the linear combination of prediction rules in the test sample. If K is large we can do the Lasso aggregation instead, where we minimize the sum of squared prediction errors of predicting Yi by the linear combination of prediction rules in the test sample plus a penalty term which penalizes the size of the coefficients.

Let's illustrate this idea in our case study. Here we consider aggregating prediction rules based on past Lasso, Elastic Net, Random Forest, Boosted Trees, and Neural Network. We estimated the aggregation coefficients using Least Squares and Lasso. From the estimated coefficients reported in this table, we see that most of the weight goes to the prediction rules generated by Least Squares on a simple model and by the Random Forest.

Other prediction rules receive considerably less weight. Moreover, the adjusted R squared for the test sample gets improved from 35% obtained by the Random Forest to about 36.5% obtained by the aggregated method. So let us summarize. We discussed the assessment of predictive performance of modern linear and non-linear regression methods using splitting of data into training and testing samples.

The results could be used to pick the best prediction rule or to aggregate prediction rules into an ensemble rule which can result in some improvements. We illustrated these ideas using recent wage data.

2.3.4 inference Using Modern Nonlinear Regression Methods. Case study: The Effect of Gun Ownership on Homicide Rates

In this segment, we consider inference for the modern nonlinear regression. Recall the inference question. How does the predicted value of Y change if we increase a regressor D by a unit, holding other regressors Z fixed? Here we answer this question within the context of the partially linear model which reads, Y = beta D + g of Z + epsilon.

Or the conditional expectation of epsilon given Z and D = 0, where Y is the outcome variable, D is the regressor of interest, and Z is the high dimensional vector of other regressors or features called controls. The coefficient beta provides the answer to the inference question. In this segment, we discuss estimation and confidence intervals for beta.

We also provide a case study in which we examine the effect of gun ownership on homicide rates. In order to proceed, we can rewrite the partial linear model in the partialled-out form as tilde Y = beta tilde D + epsilon, where expected value of epsilon times tilde D equals zero.

And where tilde Y and tilde D are the residuals left after predicting Y and D, using Z, namely. Tilde Y is equal to Y minus l of Z. Tilde D is equal to D minus m of Z, where L of Z and M of Z are conditional expectations of Y and D given Z.

Our decomposition can now be recognized as the normal equations for the population regression of tilde Y on tilde D. This implies the Frisch-Waugh-Lovell theorem for the partially linear model which states that the population regression coefficient beta can be recovered from the population linear regression of tilde Y on tilde D.

The coefficient is the solution to the best linear prediction problem where we predict tilde Y by linear functional of tilde D. We can give an explicit formula for beta which is given by the ratio of the two averages that you see in the formula. We also see the beta is uniquely defined if D is not perfectly predicted by Z, so that tilde D has a non-zero variance.

The theorem asserts that beta can be interpreted as a regression coefficient, or residualized Y on residualized D, where the residuals are defined by taking out the conditional expectation of Y and D given Z from Y and D. Here we recall that the conditional expectations of Y and D given Z are the best predictors of Y and D using Z.

Now we proceed to set up an estimation procedure for beta. Our estimation procedure in the sample will mimic the partialling out procedure in the population. We have an observations on Yi, Di, Zi. We randomly split the data into two halves. One half will serve as an auxiliary sample, which will be used to estimate the best predictors of Y and D given Z, and then estimate the residualized Y and residualized D.

Another half will serve as the main sample and will be used to estimate the regression coefficient beta. Let A denote the set observation in the auxiliary sample and M the set if observations names in the main sample. Our algorithm proceeds in three steps. In step one, using auxiliary sample, we employ modern nonlinear regression methods to build estimators hat l of Z and hat m of Z are the best predictors, l of Z and m of Z.

Then using the main sample, we obtain the estimates of the residualized quantities. Check Yi = Yi- hat l (Zi), and check Di = Di- hat m(Zi) for each i in the set M. And then using ordinary squares, check Yi on check Di will obtain the estimate of beta, denoted by hat beta one, and defined by the formula.

In step two, we reverse the roles of the auxiliary and main samples, repeat step one and obtain another estimate of beta denoted by half beta two. In step three, we take the average of the two estimates from steps one and two obtaining the final estimate at beta. Using the formula it can be shown that the four ling result quote, if estimators had l of zed and had m of zed provide approximation to the best predictors l of zed and m of zed that are sufficiently of high quality, then the estimation error in estimated visualized quantities have a negligible effect in hat beta, and hat beta is approximately distributed as a normal variable would mean beta and variance V over n for the expression of the variance appears in the form.

In words, we can say that the estimator hat beta concentrate in the square root of V over n neighborhood of beta which deviation control by the normal law. We can now define the standard error of hat beta and square root of hat V divided by n where hat V is an estimator of V.

The result implies that the confidence interval given by the estimate plus minus two standard errors covers the true value beta for most realizations of the data sample. More precisely approximately 95% of the realizations of the data sample. In other words, if our data sample is not extremely unusual, the interval covers the truth.

Given that we can use a wide variety of methods for estimation of L of Z and M of Z, it is natural to try to choose the best one using the data splitting. In the construction we described we use the auxiliary sample A to estimate predictive models using modern non-linear regression methods, we can, in principle, use the main sample, m, as the validation or test sample to choose the best model for predicting y, and the best model for predicting d, following the procedures we explained in the previous segment.

Or we can use the main sample m to aggregate the predicted models for y and aggregate the predicted models for g, using least squares or lasso following the procedures we explained in the previous segment. The previous inferential result continues to hold if the best or aggregated prediction rules are used as estimators had m of Z and hat l of Z of m of z and l of z in the algorithm we presented above.

The required condition for this is that the number of rules we aggregate over or choose from is not too large relative to the overall sample size. We provide a precise statement of the required condition in the supplementary course materials. We'll next consider the case study where investigate the problem of estimating the effect of gun ownership on the homicide rates in the United States.

For this purpose, we'll estimate the partially linear model, where Yj,t = beta D j, (t- 1) + g of Zj,t + Ej, t. Here, the outcome Yj,t is a log of the homicide rate in the county j at time t. Dj, t- 1 is a log of the fraction of suicides committed It was a firearm in the county j at time t minus one, which we use as a proxy for gun ownership and Zj,t are control variables which include demographic and economic characteristics of county J at time T.

The parameter beta here is the effect of gun ownership on the homicide rates controlling for county level demographic and economic characteristics. To account for heterogeneity across counties and time trends in all these variables, we have removed from them the county-specific and time-specific effects. Let is now describe the data sources.

The sample covers 195 large United States counties between the years 1980 through 1999, giving us to 39 hundred observations. Control variables Zjt are from the U.S. Census Bureau and contain demographic and economic characteristics of the countries such as the age distribution, the income distribution, crime rates, federal spending, home ownership rates, house prices, educational attainment, voting patterns, employment statistics, and migration rates.

As a summary statistic, we first look at a simple regression of the outcome on the main regressor without controls. The point estimate is 0.28 with the confidence interval ranging from 0.17 to 0.39. This suggests that gun ownership rates are related to gun homicide rates. If the gun ownership increases by 1% relative to trend, then the predicted gun homicide rate goes up by 0.28% without controlling for counties' characteristics.

Since our goal here is to estimate the effect of gun ownership after controlling for a rich set of county characteristics, we next include the controls and estimate the model by an array of the modern regression methods that we have learned. We present the results in a table. For the first column shows the method we used to estimate M of Z and L of Z.

The second column shows the estimated effect and the final column shows 95% confidence interval for the effect. The table shows the estimated effects of lagged gun ownership rate on the gun homicide rate, as well as the 95% confidence bands for these effects. We first focus on the Lasso method.

The estimated effect is about 0.25. This means that a 1% increase in gun ownership rate as measured by the proxy, this predicted quarter percent increase in gun homicide rates. The 95% confidence interval for the effect ranges from 0.12 to 0.36, similar point estimates and confidence intervals obtained by the Least Squares method.

The Random Forest also gives similar estimates, however, the confidence interval for this method is somewhat wider, covering the range from 0.7 to 0.42. Next, in order to contract best estimate of beta, we evaluate the performance of predictors had L of Z sets and had M of Z estimated by different methods on the auxiliary samples using the main samples.

Then we pick the methods giving the lowest average, MSE. In our case, ridge aggression and Random Forest give the best performances in predicting outcome and the main regressor respectively. We then use the best methods as predictors and estimation procedures as described above. The resulting estimate of the gun ownership effect is 0.24 and is similar to that of Lasso.

And the confidence interval is somehow tighter, now ranging from 0.13 to 0.35. Let us summarize. In this segment, we have discussed the use of modern nonlinear regression methods for inference, the procedure relies on sample splitting in order to avoid overfitting which made it hard to control theoretically. We applied these inference methods to the case study, where we estimated the effect of gun ownership rates on the gun homicide rates in the United States.

2.4 Causality and Regression

2.4.1 Causality. When It Can and Cannot be Established From Regression

In the previous lectures, we learned about the classical and modern regression methods and their uses for prediction and inference. This segment is about the use of regression to infer causal relations and answer causal questions. The causal questions are important and they arise in many real-world problems, especially in determining the efficacy of medical treatments, social programs and government policies, and in business applications.

For example, does a particular drug cure an illness? Or, do more lenient gun laws increase murder rates and other types of crime? Or, does the introduction of a new product raise profits of a company? We begin by noting that regression uncovers correlation or association between outcome variables and regressors.

However, correlation or association does not necessarily imply a causal relation unless certain assumptions hold. The association between outcome variable Y and a regressor D formally means that D predicts Y, namely the coefficient in the linear regression of Y when D is not zero, or that, more generally, conditional expectation of Y given D is not constant.

This is what the regression analysis gives us. So why doesn't association between Y and D necessarily imply that D causes Y? Here is a contrived example. Suppose we conduct an observational study on people's us of pain-killers and pain. We record the outcome variable Y, which is whether a person is in pain, and a regressor variable D, which is whether the person has taken a pain-killer medicine.

It's likely that people will take these pills when they are in fact in pain and so we are likely to find that D indeed predicts Y so that the regression coefficient of Y on D is positive. Y and D are associated but D does not cause Y as we know from the clinical trials in medicine that establish that pain killers actually work to remove pain.

An ideal way to establish a causal relation from the regression analysis is to rely on data from a randomized control trial. And in the mass control trials we have the so called random assignment or. Remember the treatment variable T is a sign or generic D that is independent of specific outcomes.

Specifically in randomized control trials or experiments we have a group of treated participants and the untreated ones. The letter called the control group, and there are no systematic differences between the two groups when the trial starts. After the trial is completed, we record the outcomes of interest for participants in the two groups, and carry out the regression analysis to estimate the regression equation Y = alpha + beta D + u.

Here the regression coefficient beta D measures the causal input, or the treatment, on average outcomes. And so beta is called the average treatment effect. Under random assignment we don't need to use any additional regressors or controls. However we may use additional controls to improve the precision of estimating the average treatment effect better.

This is called the Co variate adjustment method. Specifically we may set up a linear or partially linear model Y equals beta G plus G of zed plus epsilon and carry out inference variable using the inference methods that we have learned in this module. The randomized control trials represent the golden standard in proving that medical treatments and social programs work, and for this reason they are very widely used in medicine and in policy analysis.

And randomized control trials are also widely used in business applications, under the name of AIB testing. To evaluate whether new products and services raise profits. But if we don't have access to the data from the randomized controlled trials. What if we work a data set that is pure observation.

Under what conditions can we claim that we've established a causal relationship. A sufficient condition is that D, variable of interest is generated as if randomly assigned, conditional on the set of control Z. This is called the conditional random assignment, or conditional exogeneity. There is condition on Z, variation in D is as if it were generated through some experiment.

As in a randomized control trial. In this case, we can also apply the partial linear regression model, Y = beta D + g(Z) + epsilon, and beta D indeed measures the causal effect of D on average outcome controlling for Z. We then apply the inference method that we have learned to beta, and construct confidence intervals.

One note that we would like to make here is that under conditional random assignment, or conditional unlike under pure random assignment, controls must be included to ensure that we measure the causal effect and not something else. Under pure random assignment, we do measure the causal effect and not something else whether or not we include the controls.

So on the conditional random assignment or conditional exogeneity, regression doesn't cover causal effects. For example, recall our case study of the impact of gun ownership on predicted gun homicide rates. We did find that there that increases in gun ownership rates lead to higher predicted gun homicide rates, after controlling for the demographic and economic characteristics of various counties.

If we believe that the variation in gun ownership rates across counties was as good as randomly assigned, after controlling for these characteristics, then we should conclude that the predictive effect is a causal effect. If we don't believe that this variation is as good as randomly assigned, even after controlling for all these characteristics, then we shouldn't conclude that the predictive effect we've found Is a causal effect.

Similarly, in another case study, we studied the impact of being female on the predicted wage. We did find that females get paid on average two dollars less per hour than men, controlling for education, experience, and geographical location. If we believe that the variation of gender conditional on these controls is as good as randomly assigned, then we should conclude that the predictive effect is a causal effect, which is the discrimination effect in this context.

However, if we don't believe that this variation is as good as randomly assigned, then we should not claim that this effect is due to discrimination. As you can see, making causal claims from the observational data is difficult and the success really depends on being convincing at persuading ourselves and others that the assumption of conditional random assignment or conditional exogeneity holds here, the burden of persuasion lies entirely on the data scientist, if indeed she or he is willing to make the causal claim.

I would like to conclude this module with some parting remarks. In this module we studied the use of the classical and modern linear and non-linear regression for purposes of prediction and inference, including causal inference. The material in this module is very difficult and if you've managed to navigate through it you have done extremely well.

If you are interested in having additional reading and in replicating the case studies yourself, I have the following materials for you. First I have a complete set of slides. That accompany the video lectures. And second, I have data sets and programs written in R that carry out the studies.

I encourage you to check and try those out. With this, let me say goodbye, and wish you luck in your data science adventures.

Module – 3: Classification, Hypothesis Testing and Deep Learning (David Garmanik and Jonathan Kelner)

3.1 Hypothesis Testing and Classification

3.1 Introduction

Hello and welcome to the module on classification and hypothesis testing. I'm David Garmanik from Sloan School of Management at MIT. I'm also a member of Center for Statistics at IDSS. My background is in applied probability, optimization, and algorithms. And I regularly teach probability statistics and optimization methods to Masters of Business Administration students.

I will teach this module jointly with Jon Kelner.

Hi, I'm Jonathan Kelner and I'm a professor in the Math Department and Computer Science and Artificial Intelligence Lab at MIT. My work also focuses on algorithms, optimization, and applied probability. And I regularly teach classes on mathematical foundations of probability, as well as on the algorithmic and computational techniques required to manipulate, understand, and extract meaning from data.

At the beginning, we gave just one example of a concept commonly called classification in the field of statistics and machine learning. The goal of classification is to be able to place a given object into the appropriate category based on its attributes, typically given by data associated with this object.

In the credit card example, the object is the credit card transaction nd there are two assigned categories, legitimate and fraud.

3.1.1 What Are Anomalies? What is Fraud? Spams? – Part 1

It's a beautiful sunny day and you just arrived at Schiphol Airport in Amsterdam. A sparkling European airport suggests the start of a great three day vacation. Tired from a long overseas flight you walk into a cafe and order a fresh cup of coffee with a apple Danish but your Visa card is declined.

The credit card company suspects a fraudulent transaction. And in order to prevent the fraud it stops it. You walk to the nearest ATM to get some local currency. Perhaps, only to discover that your cash withdrawal is declined as well.

A few months later, during a business trip to Toronto, you receive a phone call from your credit card company.

The operator wishes to inquire about the legitimacy of a transaction on your credit card that was attempted the previous evening, in which someone tried to use the card to purchase some liquor in New Jersey. Yesterday evening you were en route to Toronto, and you certainly did not intend to buy.

Your card was in your possession at all times, but the operator informs you that these days it's not hard for criminals to copy the information on a magnetic strip to a different card, which was probably what occurred. Luckily, the bank figured out that the purchase was almost certainly fraudulent, and it stopped the transaction.

The two examples above are really two faces of the same coin. Your credit card company constantly needs to check the legitimacy of the transaction credit cards. The verification needs to be done rapidly in a matter of a split second. It needs to approve the transactions the company deems to legitimate, and decline the ones it suspects to be fraud

In the first example, a legitimate transaction was declined by your bank, creating a major inconvenience on your end.

But in the second example, an illegitimate transaction was thankfully declined by the bank, saving you from the frustration of dealing with a fraud and possibility of even a financial loss.

Credit card companies constantly deal with assessing the likelihood that a given transaction is illegitimate. Need to minimize the risk of either declining legitimate transaction or allowing a fraudulent one. In this module, we'll introduce statistical methods that assist companies in assessing these types of risks. It'll be our first example of a very common category of questions which we'll refer to as binary classification

3.1.2 What Are Anomalies? What is Fraud? Spams? – Part 2

The example above, for the credit card transaction and fraud detection, is one of the many, many examples where a classification problem needs to be solved based on the availability of imperfect information usually in the form of some data. Another very similar example of fraud detection is medical fraud.

A medical provider sends an invoice to a health insurance company for some services provided to a patient. Sometimes unfortunately, the services are fake and possibly even the patient was nonexistent. The insurance company needs to come up with a quick judgment on the validity of the claim, honoring the claims it deems valid and declining the claims that look suspicious.

A set up very similar to the one with the credit card example. A slight difference is the timing of the decision and the actual process. If the claim is dimmed to be fraud, it is sometimes turned to a human expert who can make a further determination regarding the validity of the claim, and only upon further investigation make a final determination.

Whereas in the credit card example, the determination is conducted immediately. As before the problem is classifying the claim as legitimate versus fraud.

Let's consider a different example from the healthcare area. A patient is rushed to a hospital emergency center with severe symptoms including chest pain, shortness of breath, heart palpitations, sweat and some other symptoms usually associated with a heart attack.

The doctors needs to make a quick determination of whether this is indeed a case of a heart attack and take the appropriate measures. Again this is the example of classification. Heart attack versus not heart attack. The determination needs to be made rapidly in potently life threatening situation. All of the examples above fall into the category of so called binary classification, where the goal is to place the given object into one of two possible classification groups.

Other examples of binary classification are determining the gender, male or female of a photograph, spam versus not spam for emails, impact versus no impact for a potential drug, healthy versus cancerous for tissue in a tumor, global warming versus normal temperature fluctuations and geoscience among many others.

The history of humans facing various kinds of classification problem is very old.

Think about the questions enemy versus friend, murderer versus not a murderer. Think about any civil or criminal trial. The goal of the jury is to solve a binary classification problem. Guilty versus not guilty, based on the data, evidence. For good or for bad, the binary classification problem in the field of criminal justice is typically not solved using statistical methods per se, but using different methods such as witness testimony.

But one can argue that indirectly, historical data plays an important role here. A person possessing a weapon of murder is far more likely to be a perpetrator than the one without a weapon in similar circumstances. At the same time, of course, other factors play an important role, such as witness testimony or simply prejudice.

Think about the binary classification outcome of Salem witch trials.

Even animals routinely solve binary classification problems when they face a potential threat. Upon seeing a creature, they need to make an instant termination of the kind, predator versus not a predator, and react rapidly and accordingly. The faster and more accurately this classification problem is solved the better the chances of survival are for the animal.

In nature this classification problem is solved by the brain processing the given data associated with potential threat. Including the visual image, smell, speed of movement, etc. One artificial model based on how animals solve such a binary classification problem is called the neural network. It's intended to loosely model brain activity and it will be discussed later in this module.

3.1.3 What Are Anomalies? What is Fraud? Spams? – Part 3

The classification problems we will discuss in this module will be the kind sold by in statistical methods based on your medical data. This applies to most, but not all of the examples above. For example, the criminal justice system example relies more on witness testimony rather than hard data and such examples will not be considered in this module.

Depending on the context, one might deal with situations where the data is in abundance, or conversely, is very scarce. For example, in trying to design a classifier for spam, lots of data is typically available in the form of prior emails already categorized as spam versus not spam. Conversely, in clinical trials the data comes from experiments conducted on human volunteers.

Such things are very expensive both financially and otherwise. As a result, are based on much smaller amounts of experimental data.

But even in the case where the data in abundant, one may face a huge imbalance of the data availability in the different categories. For example, typically only a very small proportion of credit card transactions are fraud.

Thus, while the amount of data is huge, those that fall into the fraud category constitute a much smaller amount. Similarly only a tinny number of say images taken from a street camera correspond to wanted people such a criminals, terrorists or lost children. Furthermore, in trying to correctly classify the type of object the implications of misclassification actually may have very different consequences with different levels of severity.

This brings us to the subject of type one versus type two errors. When solving a binary classification problem, we encounter two types of errors. Consider again a medical fraud example. The two categories associated with this example are legitimate and fraud. The two types of errors we're about to discuss relate the ultimate classification decision made by a classifier to the crack but unknown to the classifier category.

Suppose for example an insurance company receives a claim which happens to be a legitimate one. Though of course company has no way of knowing that for sure. Suppose further that an automated system designed to make a preliminary determination declares the claim to be a fraud clearly making a mistake, somewhat arbitrary and somewhat non-creatively.

We call this type 1 error to contrast it with the following mistake naturally called type 2 error. A claim which happens to be a fraud though unknown to the classifier is received but the classifier determines it to be a legitimate one.

Consider our example of a patient with severe symptoms who is rushed to an emergency center.

Suppose this patient is not having a heart attack, even though he has severe symptoms that make the heart attack hypothesis very plausible to doctors. Again, this is clearly not something the doctors know a priori. Suppose they make an incorrect inference and decide that the patient is having the heart attack, and they conduct the treatment appropriate for a heart attack.

This is a type 1 classification error. Conversely if a patient is indeed suffering a heart attack but the medical team fails to see this we're dealing with a type 2 error.

Deciding which type of the classification is type 1 versus type 2 is clearly somewhat arbitrary. And depends on the category which is declared to be the first category and which one is the second one.

This is really up to the classifier, and simply is a matter of labeling the classes. Typically, in statistics the classes are called null hypothesis and alternative hypothesis, and we will talk about this in our future lectures. What is important to keep in mind though, is the potential difference in severity of making type one versus type two error.

For example, in the case of medical fraud discussed earlier, a type one error leads to an inconvenience for the medical provider who needs to waste time proving the validity of his claim. If this happens to often the provider may stop excepting this type of insurance which could cost the insurance company to loose customers.

On the other hand, a type 2 error, where the insurance company pays for a fraudulent claim, leads to a financial loss coming from paying for a false claim. It may also lead medical providers to believe they can get away with fraud and embolden them to commit further fraud in the future.

It's a judgment call as to what type of error is more severe, and thus, which one needs to be mitigated more stringently. But clearly a balance needs to be struck. Contrast this with the implications of the two types of errors in the heart attack example. Type 1 error leads to a healthy patient or a patient suffering from something much less severe then a heart attack, receiving the treatment for a heart attack.

Conversely, type two error results in a patient suffering from a heart attack Not receiving the right treatments. The potential implications of a type 2 error are very severe, possibly leading to death. Whereas the implication of a type 1 error, while possibly very unpleasant, are usually not life-threatening.

The potential difference of severity of two types of error is important to keep in mind since, as we will see later, there is a fundamental trade off between the likelihood of two types of errors.

With a fixed amount of data, one can typically make type one errors small only at the expense of increasing type two error and vice versa. Balancing the two types of errors should depend ultimately on the severity of the implications of these errors. Think about the implications of type one and type two errors for our other examples.

Guilty versus not guilty. Predator versus not a predator.

A parting thought we would like to discuss in this introductory lecture is the importance of a solid scientific approach in applying a variety of classification methods, such as the ones that we are going to be discussing in future lectures.

While these methods are based on rigorous statistical foundations, it's easy to misuse them and to obtain wrong conclusions by misapplying them as such. Such misuse is particularly likely in instances where the party conducting the classification analysis has a stake in a particular outcome. For example, a pharmaceutical company testing the effectiveness of a particular drug or scientists testing a particular scientific hypothesis typically wish for one of the two particular outcomes.

Understanding how to recognize and avoid such bias is a crucial yet surprisingly subtle question. At the end of this module we'll discuss some frequently occurring examples of the misuse of classification methods. And we'll discuss ways to prevent them.

3.2 Binary Classification: False Positive/Negative, Precision / Recall, F1-Score

In the previous section we introduced binary classification, in which the goal is to put an object into one of two possible categories. Our aim later in this module will be to design effective methods for performing this task. But in order to do this, it's important to have a well defined notion of what it means for a classifier to be effective.

In this video we'll examine some of the considerations that go into evaluating the quality of a classifier and I'll present a few of the metrics that are frequently used to do so. As a concrete example, let's consider the case we described in the last video where a credit card company is trying to identify fraud.

When presented with a transaction, the classifier has to either label it fraud or not fraud. We discussed two different ways that the classifier could make a mistake. A type I error, where we identify a legitimate transaction as fraud, and a type II error, where we label a fraudulent transaction as legitimate.

It's useful to think of the classification problem as answering the yes or no question, is this transaction fraud? So that we can then describe the categories as positive, corresponding to the answer, yes, this is fraud, and negative, corresponding to the answer, no, this is not fraud. It's often helpful to arrange the possibilities as a table, where the columns correspond to the answer the classifier gives, and the rows correspond to the true underlying answer.

We thus have four possibilities. The first is that the transaction was fraud, and the classifier correctly labeled it as fraud. In this case the true category was positive and the label was positive as well. We'll call this a true positive. The next possibility is a type I error.

Here, the transaction was not fraud but the classifier incorrectly said that it was. In this case, the true category was negative, but the classifier falsely labeled it as positive. So we call these false positives. The third possibility is a type II error, where the transaction was fraud, so the right answer was positive, but the classifier falsely labeled it as negative.

So we call these false negatives. The final possibility, which hopefully is the most common, is that the transaction was legitimate. So in this case, the true answer was negative, and the classifier correctly labelled it as such. We call these true negatives. We can then visualize the performance of the classifier by writing in each box the number of times that the corresponding possibility occurred, forming what is sometimes referred to as the confusion matrix.

Note that the upper left and lower right numbers correspond to correct answers and the upper right and lower left ones correspond to mistakes. Now, how should we evaluate the quality of our classifier? A natural first guess would be to just look at what fraction of the time it gives the right answer.

However, this number can be misleading, particularly in settings like this one, where one of the categories is much more common than the other. To see the problem, suppose that 1% of the transactions are fraudulent. And suppose that the credit card company's very lazy fraud detection department just wrote a computer program that labeled everything as legitimate.

This is clearly not a very helpful classifier, since it never says there's any fraud. But it's correct 99% of the time. To get a better evaluation, we need an approach that provides more nuanced information by the different ways in which the classifier can be right or wrong. A common way to do this is to look at two numbers, precision and recall.

Roughly speaking, precision will tell us how often the classifier is right when it says something is fraud. And recall will tell us how much of the actual fraud that occurs we correctly detect. Let's now define these mathematically. We define the precision to be the number of true positives divided by the number of true positives plus the number of false positives.

Note that the denominator equals the total number of examples that the classifier labeled as positive. This number tells us what percentage of the time an instance labeled positive was actually positive. In our fraud example, the classifier labeled 2 plus 8 equals 10 instances as fraud. Of these, 2 were actually fraud, so the precision was 2/10 = 0.2.

We then define the recall to be the number of true positives divided by the number of true positives plus the number of false negatives. Here, note that the denominator equals the total number of examples in which fraud actually occurred. This number tells us what percentage of the actual positive examples our classifier detects.

In the fraud example, there were 2 plus 4 equals 6 instances of fraud, and the classifier correctly caught two of them. So the recall here was 2/6, which is 1/3. This number measures how sensitive our classifier is to indicators of possible fraud, so it's sometimes called the sensitivity.

Note, this would actually catch the problem with the lazy fraud detection algorithm that we described before. The one that describes everything as legitimate, since in this cases the sensitivity would be zero. Know that there's often a trade-off between these two quantities. A more aggressive fraud department that flags a huge number of things as fraud, sometimes incorrectly, would have a high recall, since it would catch a lot of the fraud that occurs.

But it would have a low precision, since it would also flag a lot of legitimate transactions as fraud. On the other hand, a very conservative department that only flags the most obvious cases would probably have high precision. But it would miss the subtler cases of fraud, and would thus have lower recall.

How one trades off between these two numbers is a judgment call, based on the specifics of the situation. It is often most informative to look at both of these numbers separately, since they measure different properties of the classifier. However, if you want a single number to evaluate the quality of a classifier, there's a statistic called the F-score, or sometimes called the F1 score, that combines the two.

Our first attempt at combining the two numbers would be to just take their average. However, this often isn't quite what we want. In particular, suppose we are in the case where the classifier labels almost everything as positive. Here, the recall would be great, that is very close to one, but the precision would be quite small, and thus close to zero.

This would probably not be a very good classifier, but averaging the two numbers produces something close to a half. Instead, we would like a way of combining the two numbers that puts more emphasis on the smaller value. A good choice for this turns out to be what's known as the harmonic mean.

The harmonic mean of two numbers, x and y, is given by 1 over the average of 1/x and 1/y. This will always be in between x and y, but it will be small if either of the two numbers is small. So it's closer to the smaller of the two numbers, in the arithmetic means.

The F-score is then defined to be the harmonic mean of the precision and the recall. So it equals 1 over the average of 1 over the precision, and 1 over the recall. Simplifying this gives a slightly nicer expression. F1 = 2 x precision x recall over precision + recall.

So in summary, we now have a couple of measures for evaluating how good a job a classifier does. We defined precision and recall, two numbers that capture different facets of the quality of classifier. And then we showed how to combine them into an F1 score. This gives us a set of statistics we can use so that later we can decide, when we have some classifier, how good a job it's doing.

In the next section, we're gonna switch a little bit and talk about hypothesis testing. And then we'll come back to how to build effective methods for constructing specific classifiers.

3.3 Hypothesis Testing

Today, we will introduce a very important statistical concept. Hypothesis testing. The meaning of the term hypothesis testing is precisely what it says. Using a data observed from a distribution with unknown parameters, we hypothesize that the parameters of this distribution take particular value and test the validity of this hypothesis using statistical methods.

We test a hypothesis. The method of hypothesis testing has many applications. Say a new drug for lowering blood pressure was just produced by a certain pharmaceutical company. Does the drug has a meaningful effect on lowering the blood pressure as the company claims? Is the impact larger than that of the competitive drugs.

A community organization claims that the police in a certain precinct is prejudiced and tends to stop disproportionally many people of color during traffic patrols. They have produced some data to support their claim showing much larger number of people of color stopped than say, on average, across US. A car manufacturer claims to have achieved a significantly lower emission rate than those of its competitors and has provided data to support their claim.

The data supporting these kinds of claims is naturally subject to a lot of uncertainty. And one can never be completely certain whether or not the claim has any validity to it. One has to rely on statistical methods to draw such conclusions. Today, we introduced some of this methods.

3.4 Confidence Intervals

To get started, we should review some key concepts related to the, so called, confidence intervals. The confidence intervals are intended to provide us with probabilistic level of certainty about our conclusions, regarding parameters of some probability distribution. Let us introduce it somewhat more formally. Suppose you have several observations, X1, X2, and so on, Xn from the probability distribution.

Suppose you have a prior knowledge that this is a normal distribution, but with some unknown mean value, mu. Supposed at the same time you happen to know the standard deviation, sigma, for this normal distribution. This is not as unreasonable as it may sound. As you may recall, we denote such distribution as follows.

Our goal is to estimate mu, a natural estimation for mu is simply the average of our observations, which we denote by x bar, like this. Naturally, the actual value of X bar, practically, will never equal to the mean value mu. Our hope is, though, that the estimation is not too far.

But, how far is too far? This is where the concept of confidence interval helps. Let us use the property of the normal distribution, which says, that in fact, X bar itself also has a normal distribution, with the same mean value, mu, and standard deviation, equal to sigma over square root of n.

Namely, the standard deviation of X bar is this. Using our notational convention, X bar then has the following distribution. It is very important to understand, that while X bar is intended to estimate the unknown mean value, mu, it is actually a random variable with this distribution. For such a random variable, we can compute the probability that it falls in a certain range.

For example, we can compute the probability that it does not exceed the actual, but unknown to us, mean value of mu by, say, two units. Namely, our estimation is now two units greater than the actual mean value. Here's how we compute this probability. First we write it like this.

And now we rewrite it like this, and now we rewrite it further, in the following, admittedly, weird way. Why in the world we would want to do something like this? The answer is that, what we have on the left-hand side, namely this expression, is actually a standard normal random variable.

That is, it is the normal random variable, with mean 0 and standard deviation equal to unit. Using our notation, this is, to convince yourself of this, you need to recall some of the basic properties of the normal random variable. Like, what happens to it when you multiply it by some number, and observe that the mean of X bar is mu, and the standard deviation is sigma divided by the square root of n?

Namely, recall that X bar has distribution as follows. The second part of the magic, is that the expression, which you see here, magically does not involve the unknown value mu. It involves sigma, which we know, and the sample size, n, which we certainly know, since this is just a sample size.

So we just need to compute the probability that a standard normal random variable does not exceed the following value. But this we can do, either by consulting the so-called standard normal distribution table. Search Google for this, for example, or applying any standard statistical packages. Let us do an example.

Say you have collected a data set consisting of 40 observations. In this case, the sample size, n, is 40. Suppose you happen to know that the standard deviation of this distribution, from which the sample is generated, is 6.8. From this, we can immediately find that, from the standard normal distribution table, we'll find that the probability that the standard random variable does not exceed this value, is approximately 0.968.

You have computed the average of these 40 observations, and let us say you found it to be 4.7. Then according to our derivation, the true, but unknown, mean value mu, is at least 4.7 minus two, which is 2.7, with probability 0.968. We also say that our confidence level of this statement is 0.968, or roughly 96%.

In this example, we have assumed the knowledge of the standard deviation sigma. Usually in practice, we do not have it, but we can still estimate it from data using the following expression, which we denote by sigma bar. Here, X1, X2, and so on, Xn are your observations. Just like X bar is perhaps the most reasonable estimation of the actual, but unknown, mean value mu.

This formula gives perhaps the most reasonable estimation for true, but unknowns standard deviation Why? This is outside of the scope of this lecture, but I invite you read about this in one of the statistics books. This substitution appears to rely on circular reasoning. Shouldn't we have confidence intervals for sigma bar first, before we use it for our computation of confidence intervals for the mean value mu?

The answer is that, when the sample sizes are reasonably large, usually at least 30, the estimation for sigma bar is far more accurate than the level of error we will encounter for mu. So, this substitution is a reasonable one. There is a deeper theory behind this, which relies on the so-called, t-distribution, which we'll skip today.

In the example above, we have obtained an estimation from mu, which is only one sided, namely, it is at least 2.7. Ideally, we would like to have an estimated from above and from below. Additionally, often we want to have a target level of confidence level, such as 0.95 or 0.99, rather then whatever comes out here, like 0.968.

This can be done by, sort of, reverse engineering our interval, so that the resulting certainty equals the target we need. Say we want to have confidence level of 0.95. Here's how we can find the corresponding confidence interval. The derivation here is terse, and I invite you to take some time to parse it, or consult a statistics book for a more detailed explanation.

But it is worth trying to parse it yourself first. So our goal is to find a value c, such that the following is true. We can rewrite this equivalently as follows. And, if you find the right value for c, for this to be the case, our 95% confidence interval would be of the form X minus c, to X plus c.

Now saying that this relationship is true, is the same as the following being true, which is the same as the following being true. From the standard normal distribution table, I can find the magic number, 1.96, which satisfies the following property. We obtain that the following must be the case, from which we find the formula for c as follows.

Since we know the standard deviation, which in our example was 6.8, and since we know the sample size, which in our case, is 40. We can compute c as follows, we conclude that the true, but unknown, mean value mu lies within approximately 2.1 units from X bar, which is 4.7 with confidence 0.95, namely 95%.

This interval is in fact defined by 2 values, the lower value here, and the upper value here. We can summarize our conclusions as follows, the true, but unknown, mean value mu is at least 2.6, and at most 6.8, with 95% confidence. The interval length might appear as too wide.

This is perhaps true, but sometimes even a very wide interval can provide an extremely useful information. When, for example, you want to figure out whether the mean value equals 0 or not. That is, you hypothesize that the mean value is 0 and you want to test it. In other words, you want to conduct a hypothesis testing.

3.5 Hypothesis Testing: Validity of Binomial Distribution

In the discussion of hypothesis testing, we have considered only one example involving normally distributed random variables. But the hypothesis testing method is much broader than that, and allows you to test the validity of the probability distribution itself. Let us illustrate this in a special case when the underlying distribution is binomial.

In the beginning of the lecture, we mentioned an example of using hypothesis testing for the analysis of police racial profiling and possibly testing the hypothesis that the police is particularly biased against people of color when they pull over drivers on the road. Let us say that in the year of 2015 among the drivers stopped by the police 35% were black drivers.

The actual figures are somewhat different and you can find some of these figures on the Bureau of Justice Statistics website. One way to think about the statistics is that if one draws a random driver from the list of all drivers stopped by the police in the year of 2015, there's a 35% chance that this driver is African American.

Since this percentage is higher than the percentage of black drivers among all the roughly 200 million drivers in the US, this already suggests a bias. But let us imagine the following situation. Community organizers in several towns across several states, claim that in fact the situation is worse in their communities than it is in the US on average, however disturbing these averages are on their own.

They claim that even if one accepts for the sake of argument that racial profiling is non-existing from a statistical point of view, in the US on average the police precincts in their towns do rely on racial profiling. To support their claim, the data was provided for two precincts, which we just call Precinct A and Precinct B for simplicity.

The data shows that during the year of 2015, out of approximately 12,567 drivers stopped by the police, in fact 4,513 were black. 4,513 divided by 12,567 is approximately 0.359, which is larger than 35% and hence the claim of racial profiling. For the second Precinct B among 15,687 drivers stopped 5,562 were black.

Again, 5,562 divided by 15,687 is approximately 0.354, which is also larger than 35% and the claim of racial profiling is made again. How can we asses the statistical validity of these claims? We do this by conducting the hypotheses test. We take as our null hypothesis, namely the hypothesis to be tested, that the drivers pulled over by police in these precincts are as likely to be black as for the statistics across US suggest.

Namely, a driver pulled over by police in these precincts is black with 35% likelihood, as is the national figure. We set Type 1 error threshold alpha to be 5%, this is up to the person conducting the test, of course and different error rates may be considered, but let us accept if for now.

Assuming the validity of this hypothesis, we will compute the likelihood that among 12,567 drivers in Precinct A at least 4,513 were black. Namely at least as many as was the number of black drivers actually pulled over. If this probability is greater than 5%, we accept the hypothesis as valid with 5% error rate and say that the exceedance of 35% rate in this precinct is just a statistical fluctuation.

If it is less than 5%, we will reject this hypothesis and say that the number of black drivers actually pulled over was unusually high. We will conduct a similar analysis for Precinct B. How can we compute these probabilities? If the 35% hypothesis is true, then according to the binomial distribution, the likelihood that out of 12,567 drivers at least 4,513 were black can be computed using the following formula for binomial distribution as follows.

Take some time to convince yourself that we have applied the formula for the binomial distribution correctly. This might look horrendous but in fact most statistical softwares and many other non-statistical packages such as MathLab will find and compute this expression in a split sec. The answer in this case is 0.016 or about 1.6%.

We conclude that under the null hypothesis, no profiling. The likelihood that out of 12,567 stopped drivers 4,513 were black is approximately only 1.6%. Since this is below 5% selected for Type 1 error tolerance, we must reject the hypothesis. Thus, according to the data, it is unlikely, only 1.6% chance, that the black drivers were stopped at the same rate as in the US on average.

This of course does not legally prove the presence of racial profiling, just indicates the statistical evidence for it. What about Precinct B? Using a similar analysis the likelihood that out of 15,687 stopped drivers at least 5,562 are black is given by the following expression, which is evaluated to be approximately 11.4%.

Since 11.4% is higher than 5%, we must accept the null hypothesis. Namely, according to the statistical analysis, there is no evidence of racial profiling in Precinct B. Of course again, this is just a statistical analysis. In the reality, such an analysis might suggest a further investigation where profiling appears to take place according to the statistical evidence.

As far as the analysis can either confirm the presence of profiling or not find any evidence of it. The statistical tools can simply assist in the identifying, relatively quickly, precincts where there is some statistical evidence of profiling. From the methodological perspective, we just gave an example of testing your hypothesis regarding the validity of a certain distributional assumption, binomial distribution being the case.

One can naturally apply it to other types of distributions, such as uniform, geometric, Poisson distribution and many others. The method is very general. It is also very useful, I hope I have convinced you of that.

3.6 P-Value

In discussing the hypothesis testing method, we have introduced in particular, a method for accepting or rejecting a hypothesis which is selected to be the null hypothesis. The method was based on introducing some measurement of the observed sample such as its sample mean and computing the probability that this measurement falls within some declared region such as for example, an interval around the mean which is claimed by the null hypothesis.

The life of the interval was engineered so that the probability of falling outside of this interval, which as you may recall is called type 1 error, achieves the target value alpha. Usually taken to be, for example, 5% or 1%. In a case that this measurement indeed falls outside of the interval, the hypothesis is rejected otherwise, it is accepted.

Thus, if the hypothesis is true the likelihood it is rejected equals value alpha. There is a certain limitation of this method. This mechanism provides an acceptance rejection rule for the hypothesis, but does not provide any sense of the strengths of the measurement in either supporting or rejecting the hypothesis.

If we were to have two measurements of the sample and say based both of them the hypothesis should have been rejected, which one has stronger evidence for rejection? To gauge the strength of evidence for either supporting or rejecting a hypothesis we introduce now a new and important concept of p-value.

A formal definition of the p-value sounds something like this. The p-value is the probability of observing an outcome which is at least as hostile or adversarial to the null hypothesis as the one observed. Did you get this? Probably not. So let us discuss an example. Let us recall our example of a manufacturing device lifetime.

Recall, that the null hypothesis in this particular example stated that the mean lifetime of the manufacturing device is 9.4 years. Then we computed a magic number 0.396 which had the property that if the computed sample mean is within 0.396 lifetime units from 9.4, then the hypothesis is accepted.

Otherwise, it is rejected. Suppose you have actually gathered two samples, one with 50 elements with a sample mean equal to 8.96, and another one with 60 elements, with sample mean equal to 9.81. First note that according to each of this measurement the null hypothesis should be rejected. Verify this on your own.

Each of this measurements will be associated with some p-value which we now describe how to compute. Take first 8.96 sample average. What is the probability that when we generate a different and independent sample average of 50 observations we get the value less than 8.96 if the null hypothesis is true.

Before we answer this let us discuss why we want to compute something like this? We want to have a sense of how unlikely it is to obtain a sample average of 8.96. Now the probability for observing exactly 8.96 is practically zero. So instead, we're asking to compute the likelihood of observing something even worse than 8.96.

And this is our approximate measurement of the unlikelihood of seeing the number as far from the claim mean of 9.4 as 8.96. Now worse can mean two things. One is natural getting a number smaller than 8.96. But there is a complimentary worse, the difference between 8.96 and 9.4 is negative 0.44.

Then getting a sample of 9.4 plus 0.44, namely 9.84 is as bad as the one of 8.96. So getting a measurement above 9.84 in this sense is also worse than getting the number 8.96. Now let us compute the probability of getting a measurement either below 8.96 or above 9.84.

I say that this probability is the probability that the standard normal variable which we denote by capital Z here. Takes at most the following value for the sigma equals to 1.43 is a standard deviation of the device under the null hypothesis. I'll let you figure out why this is the case.

On your own we have done similar deviations already several times, so you should be able to do that yourself. From the standard normal distribution table we find that the probability that the standard normal rounding variable x value below a negative 2.175 is about 0.015 which is about 1.5%.

So our answer is twice that much namely 0.03 or 3%. We conclude that the p-value for our sample is 3%. Inwards, the likelihood of getting a sample which is as far from the hypothesis as the one observed is 3%. Since the p-value of 3% is kind of small, kind of.

We will be inclined to reject this hypothesis. Notice that there is a simple relationship between the p-value and our acceptance, rejection rule say based on the type one error alpha equal to 5%. The hypothesis is accepted if the p-value exceeds 5% and rejected if it stays below 5%.

A similar rule applies if the value of the type one error is said to be 1%. Often the p-value is defined as probability of being worse than observed only in one direction, in this case being below 8.96. So in our example because the normal distribution is symmetric about its mean, the actual p-value would be only half of the 3% corresponding to this probability namely only 1.5%.

The one-sided version of the p-value is particularly useful when the distribution is not symmetric around its mean as we are about to see in our next example. It is important to be explicit in what the p value stands for when conducting a statistical test. So far it seems like we have not done anything new here.

But wait a minute, we have the second sample of 60 observations with average 9.81. The difference of 9.81 and 9.4 is 0.41. As a result, the p-value is computing using the following formula. From the standard normal distribution, we find the answer to be twice 0.013, namely 0.026, that is 2.6%.

We conclude that the p-value for the second sample is only 2.6%. It is still below the 5% cutoff so the rejection is called for, but its value of 2.6% is even lower than the value of 3% for the value of the first sample. So the p-value of the second sample provides, so to speak, even stronger evidence against a null hypothesis.

If we had two samples, one with say, p-value of 11% and one with say, 18%, the null hypothesis would be accepted based on both. But arguably, the second p-value gives us stronger evidence supporting the hypothesis. So it is natural to call p-value the strength of the test, a sample mean in our case.

The larger the p-value, the stronger is the evidence supporting the hypothesis. The smaller is the p-value, the stronger is the evidence against the hypothesis. Now let us turn to our other example, testing the binomial distribution in the context of racial profiling. Recall that in precinct A, from 12,567 drivers stopped by the police, 4,513 were black.

Whereas the national figure is approximately 35%. We have computed that under the null hypothesis stating that the likelihood that a random driver stopped by the police has 35% chance of being a black driver, the probability of having at least 4,513 black drivers is 1.6%. That was below 5% selected value for alpha as a type one error and the hypothesis was rejected.

Since the binomial distribution is not symmetric. So it is approximately symmetric since it can be approximated by the normal distribution. Let us compute the one sided p-value namely the probability of observing an outcome even more hostile to the null hypothesis than the one observed. Guess what? We have already computed it.

It is 1.6%. Indeed, since the observed value was 4,513, then the p-value is the probability of having at least 4,513 stopped drivers to be black, which we have computed to be 1.6%. For the precinct B, the one-sided p-value is the probability that at lease 5,562 drivers out of 15,687 were black, which we have computed to be approximately 11.4%.

Then the p-value for this experiment is 11.4%. There is a lot of discussion as to what extent one should rely on p-values of say 5% or 1%. Type 1 errors for accepting or rejecting hypothesis in various fields. These are just useful tools and they need to be used with extreme care and never applied blindly.

The context of the underlying statistical estimation should almost invariably guide the application of the math. In our last lecture in this module, we will show examples of how the application of methods such as hypothesis testing and p-value can be misused, and lead to wrong conclusions. And I finish this part of the lecture by offering you the following challenge.

The p-value is naturally a random variable, as it is the probability of a observing a value smaller than the observed sample average. Since the simple average is a random variable so is the probability of seeing the value below it. What is the distribution of this random variable? Well, there is a magic answer.

The distribution of the p-value is uniform in the interval between 0 & 1 when the distribution corresponding to the measurement is continuous. This is not obvious at all. But try to derive this fact. It is a good brain challenge.

3.7 Misuses of Statistics

Let us say you are managing a manufacturing firm which produces a certain electronic device. And the industry standard for the lifetime of this device is to have let's say, lifetime of at least nine life years. Your engineer claims the device produced by your company fits the standard and in fact has average lifetime of 9.4 years.

And a standard deviation of 1.43 years. To support his claim he gives you sample of lifetimes for 50 devices with various actual lifetimes. In practice, one does not need to wait for ten years to test the lifetime of the device, but instead there is a procedure to estimate its projected lifetime using a certain manufacturing process.

The details of this process are besides the point in our skit. You have computed the average of the projected lifetimes for the sample provided by your engineer, and found it to be 9.6 years, which is above the industry standard of 9 years, which is good. Your goal is to test the following hypothesis.

The lifetime of the device follows a normal distribution with mean equal to 9.4 and standard deviation equal to 1.43 years. Since the mean value in this hypothesis is 9.4 which is greater than 9, confirming this hypothesis would be good news as far as meeting the industry standard is concerned.

So, our hypothesis is as follows. How can you test this hypothesis? First some notation. When you introduce and test some hypothesis, a convention is to call it a null hypothesis, and denote it as follows. Now let's do actual testing. This is done by first assuming that the hypothesis is true, selecting a certain measurement error value C.

And computing the probability that the outcome, namely the average of these 50 observations falls within C units of the claimed value of the mean value of mu. Which in our case, was claimed to be 9.4. If we choose our measurement error to be, say, 0.3 lifetime years, then we need to conclude the following probability.

How can we compute this probability? Well again, we can use the trick of manipulating x bar in the standard, normal random variable as follows. As in our discussion of confidence intervals, the random variable x bar minus mu over sigma divided by square root of n, is actually a standard normal random variable.

Let's denote it by Z for convenience. Then we have to compute the following. And we have everything we need to compute this probability. We have n, which in our case is 50. We have sigma, which in our case is claimed to be 1.43. And we have c, which we have chosen to be 0.3.

So we need to compute the following probability which we can find in the table for standard normal distribution. It is approximately 0.861 or roughly 86%. Let's summarize what we found. We have computed that when a sample of 50 observations from normally distributed random variable with mean value equal to 9.4 years, and standard deviation equal to 1.43 years is drawn.

The probability that this sample falls within 0.3 units from the mean value of 9.4 equals 0.861, namely about 86%. Wait a minute. Where did we use in this computation that our actual observed average of 50 estimated lifetimes was 9.6? This is the where we can use it to test the validity of the hypothesis.

Falling within 0.3 units of the claimed mean value of 9.4 occurs with 86% likelihood if the hypothesis is true. Our observed average is 9.6, which is in fact within 0.2 units from the claimed mean value of 9.4. Since 0.2 is in fact smaller than 0.3, we'll say that we accept this hypothesis and we do so with 95% confidence.

If you or your engineer collected a different sample with, say, average 9.05, and would want to use this average to test the same hypothesis, you would do the same thing. Compute the difference of 9.05 and the claimed mean value of 9.4, which is negative 0.35. Since negative 0.35 in absolute terms is larger than 0.3, the hypothesis should be rejected.

Notice that you have rejected the hypothesis even though your observed average was 9.05, which is higher than the industry standard of 9 years. You still reject the hypothesis because the hypothesis was about the mean of the distribution, and not meeting the industry standard per se. There's something unnerving about this method.

Whether we accept or reject the hypothesis depends on the data we observe. For different samples, we may either accept or reject the same hypothesis. But the samples come from data and this is the only thing we have access to. We can only draw our conclusion from observed data.

There is no crystal ball to tell us whether the claimed value of 9.4 years of lifetime is correct or not. In fact, there is no such thing as the correct mean value. Since the concept of the distribution underlined observed values is entirely in our head, and it is used as a convenient conceptual device to test the validity of our conclusions.

There is something else which is perhaps annoying in this example. The confidence level in our hypothesis testing example was computed to be approximately 86%. This was an implication of choosing somewhat arbitrarily the error value of 0.3 years for the lifetime duration of the device. Well this is something we can deal with.

We can choose the target level of confidence say 95% and compute the implied error value. This is very similar to what we have encountered in our discussion of confidence levels. To achieve the target value of 95%, Namely the error rate of 5%. Let us first denote this error rate by alpha.

Now we can compute the implied error value c as follows. From the table, as we recall, there is a magic number of 1.96 for the following idea. We conclude that the following is true. Looks familiar? It should. In our case, we use the fact that sigma is hypothesized to be 1.43 and the sample size is 50, and find that the value of c is 0.396 like this.

We summarize this as follows. We accept our hypothesis of mean value of 9.4, and standard deviation 1.43 with 95% confidence level if our observed sample average falls within 0.396 units from the claimed value of 9.4. Recall that the sample brought to you by your engineer had average value of 9.6 units, which is within 0.2 units of the claimed mean value.

0.2 is smaller than 0.396 so the hypothesis should be accepted. If the observed average was 9.05, as in our second example, then since the difference of 9.05 and 9.4, which is negative 0.35 is still smaller than 0.396, the hypothesis should still be accepted. It was though, rejected before.

What has changed? You've guessed it. The error value of 0.3 chosen by us was more stringent that the error value of 0.396 implied by 95% confidence level. Thus, there is something else here which is perhaps unnerving in the discussion above. The same hypothesis with the same data and thus the same average value might be accepted or rejected depending on perhaps somewhat arbitrarily chosen confidence level.

But maybe this is not so bad since it allows for choice of confidence level in different areas in different industries. For example, when dealing with drug manufacturing or anything else which relates to human lives and well being, for example airline safety. One might want to desire a very high level of confidence of let's say, 99% or even 99.9%.

Industry experts in the government do about worry about setting such standards. At the same time, say in retail industry if your goal is test the hypothesis of mean value of the number of online customers in order to see if it say significantly exceeds the mean value of the number of customers visiting your brick and mortar stores.

Making a mistake in testing this hypothesis is at worst would lead to poorer management decision. Say closing your chain of physical stores and focusing entirely on online industry, but not loss of human lives. Unless as a company manager, you feel suicidal about your poor management judgment which obviously you should not.

We have mentioned earlier the convention of calling the hypothesis, which is being tested, a null hypothesis. It is time to discuss another related concept called type 1 error, which has implicitly already surfaced in our discussion. Note than even when the hypothesis is true, say in our example the mean value was indeed 9.4 and standard deviation was indeed 1.43.

There is a chance that the hypothesis is rejected simply because the sampled average was unusually far from the true but unknown to us mean value of 9.4. In fact, we have set up our testing framework in such a way that the likelihood of this rejection of the true hypothesis equals 5%.

The error of rejecting a true hypothesis is called type 1 error, which in our case is 5%. This is contrasted with type 2 error. Type 2 error corresponds to the case of accepting by mistake a hypothesis when it is wrong. For example, accepting the hypothesis that the mean lifetime of the device 9.4, when in fact, it is not.

To estimate the probability for such an error, one naturally has to formulate the meaning of something like, 9.4 is not the right mean. Namely, one has to formulate what is called an alternative hypothesis, for example something like this. The lifetime mean value of the device is at least 10.5.

To contrast it with null hypothesis, such a hypothesis is called alternative hypothesis, and it is denoted like this.

3.8 Methods of Estimating Likelihood

On January 28, 1986, NASA Space Shuttle Challenger was scheduled to be launched from Cape Canaveral, Florida. The event was highly publicized and President Reagan was expected to make an announcement about the launch on news channels. The temperature was near freezing on this day, which was unusual for this geographic location.

Several engineers expressed concerns about the safety of the launch. Specifically they were concerned about the reliability of so-called O-rings at low temperatures, rising the possibility of failure which would lead to a catastrophe. However, there were five such rings and all five of them would need to fail for the destruction of the Challenger.

The managers at NASA, who were strongly against the postponement of the launch, noted that among the O-rings failures observed, a few happened at high temperature. Thereby conjecturing that the threat for low temperature was not worthy of postponement. After an intense discussion, they prevailed and NASA went ahead with the launch.

Tragically, soon after the launch, the Challenger disintegrated and came apart, killing everybody on board. The post-mortem analysis revealed that a failure of all five rings of the shuttle were the cause of the disaster. Was the disaster avoidable? Was there enough evidence for the engineers to convince managers at NASA that the performance of O-rings at low temperature was a serious vulnerability?

In this lecture, we discuss the methods of estimating the likelihood, namely the probability of a particular event, given the data at hand. The method has many, many applications across fields. And to illustrate the idea let us go back to some of the examples we have introduced in our first lecture.

For the credit card example, we might want to estimate the probability that the particular transaction is a fraud, given some information about this transaction. For example, the distance between the place where the sale took place and the residence of the credit card owner. Or for example, the likelihood that the claim submitted to the insurance company is a fraud, given the number of the particular procedures conducted by the medical provider.

Or the likelihood that the patient will develop a serious heart condition given his weight and average blood pressure. Or finally, going back to the beginning of this class, the likelihood that all five O-rings fail given the current temperature. This last example will be the subject of a special case discussion to be done after we introduce the basics in this lecture.

The method of logistic regression allows us to answer these types of questions by building the model of the following form. Let us discuss the details of this model. Here the symbol e, stands for the hopefully familiar Euler's constant, which roughly equals X, corresponds to an independent or explanatory variable.

Such as for example, the distance between the sale and home location for the credit card example. Or the weight of the patient for our second example. Letter E corresponds to the event of interest, such as, transaction is a fraud or that a given patient develops diabetes. The expression, which we read as the probability of the event E conditioned on the value X, is the quantity of interest.

This is what we want to estimate. Finally, the coefficients beta-zero and beta are called the regression coefficients and are to be estimated from the data. For now don't worry about how this estimation is done. We will explain it later. Consider our diabetes example. Suppose, based on the past medical data the coefficients beta-zero and beta are estimated to be as follows.

Then according to this model, a patient with a body weight of say, 135 pounds, which is by the way my weight, has a chance of developing a diabetes equal to namely approximately 13%. Incidentally, approximately 9% of US population are suffering from some form of diabetes, so actually this number is not too unreasonable.

On the other hand, for a significantly obese patient with weight, say, 275 pounds, the corresponding likelihood is estimated to be as follows. Namely, approximately 23%. According to this model, obesity is a significant risk factor for diabetes. One has to be extremely careful in interpreting this model. A more accurate way to express this conclusion of the modal would be to say that within the group of people with otherwise similar characteristics, a patient with body weight 275 pounds has roughly 23% chances of having a diabetes.

This is very similar to the care needed in discussion of conclusions of the usual linear regression model. In fact, as you probably see, the logistic regression has also strong similarity with linear regression model. Reviewing a linear regression model is probably a good idea before continuing. But strictly speaking it is not necessary at this point.

Let us go back to the expression here. Since the numerator is smaller than the denominator, the ratio takes value between 0 and 1, and therefore it is a suitable expression for a probability value. The function of the form that you see here is called logistic function. And this is where the logistic regression derives its name.

In this example, we have used only one independent variable, body weight, to estimate the probability of developing a diabetes. Naturally, some other factors might play a role here. For example, genetics. Consider, for example, the following logistic regression model. In this example, we have two independent variables, X, which as before stands for the body weight, and Y, which counts the number of parents with diabetes, and which takes value of either 0 or 1 or 2.

The coefficients beta-w is we denoted by simply beta in the previous example, though it's actual value might be different. And the coefficient beta-g is a new coefficient corresponding to the newly introduced explanatory valuable Y, corresponding to the genetic predisposition for diabetes. Let us assume that using past data, the coefficients are estimated to be as follows.

Then a patient with a weight of 215 pounds and none of the parents suffering from the diabetes has likelihood of developing diabetes equal to roughly 18%. At the same time, a patient with the same weight of 215 pounds who has both parents suffering from diabetes, according to this model has likelihood of roughly 25% for developing a diabetes over the lifetime.

We see that according to this model, the genetics plays a significant role in the likelihood of developing this sickness. Of course, this is only likelihood. While we cannot change our genetics, well, not yet, we can certainly impact other important risk factors, such as weight, diet, lifestyle, to name a few, which are known to play a role in the likelihood of developing this sickness.

Again, if one has access to further data, just as in the case of linear regression, additional independent variables can be used for likelihood estimation with better accuracy.

3.9 Support Vector Machine: Non-Statistical Classifier

The methods for binary classification that we focused on so far, work in what I'd like to call a statistical setting. Given some input to classify, they postulate some class of probabilistic models, fit various parameters. And using these they give us an estimate of the probability that the input falls into one category or the other.

If we think back to the description of binary classification, we were just required to pick one of the two categories. So trying to give a numerical estimate of the probability for each category is actually a little more than we were originally asked for. Of course these problems are very related to each other and it can't hurt to know this extra information.

But finding it sometimes comes with a cost. For instance, it may require stronger assumptions about the distribution of the data, like normality or independence. And it may limit your flexibility in choosing your methods, or require you to solve more complicated, computational questions. What I'd like to talk about now are some non statistical methods.

But just try to answer the original question of choosing one of the two categories. And don't try to fit a statistical model or assigned probabilities. I should mention that sometimes these procedures can be turned into statistical ones. But it's not always obvious how to do so, and it often involves making further assumptions.

Perhaps most importantly, the non statistical perspective suggests methods that we might not come up with from a statistical point of view. And it's led to some techniques that have been extremely effective in practice. Before launching into the methods, it's helpful to spend a little time carefully defining our question.

We'll set the problem up in the framework of supervised learning that was discussed earlier in the course. The first step here is to describe our data. To do this, we'll represent each object be classified with a list of numbers that describe it which we'll call a feature vector.

For instance, if we're trying to classify an image, say trying to determine whether it includes a human face. The feature vector may be red, green, and blue values for each pixels. If we're trying to determine if the patient is having a heart attack, the vector may comprise their blood pressure, their heart rate.

A zero or one indicating if they're conscious, and a number between one and ten indicating their self assessed level of chest pain. Now there's a lot of discretion in choosing this representation. I could give you many different ways of describing an image, or many sets of numbers that would describe a patient.

And we'll see that actually how we choose this representation is crucially important in determining the quality of our classifier. It's easy to see intuitively why a good representation should be important. Imagine that you're trying to decide, without a computer, if a picture contained a face. Now suppose that instead of giving you the picture in it's original form, I gave it to you with the pixels reshuffled in some arbitrary order.

If I told you of the order that the pixels were in, this would contain the same information as the original image. But you would obviously find the problem a lot harder. For now, you should think of the good representation as being one where the number has captured the important information in the problem.

Encode it in a nice way, and don't include too much extraneous junk. Once we choose how to represent objects with feature vectors, we can think of the objects to be classified as points in some possibly high dimensional space. The classifier is then just a division of the points in the space into two pieces according to the category it assigns them.

We can think of this as a function that takes points as input and produces a category label as its output. In supervised learning, the process is that we are given some training data which comes in the form of a collection of points that are correctly labeled as positive or negative.

Using these, we try to construct a good classification function. We are then given new points, and we classified them by applying our function. The question then comes down to how we choose this function, which is what distinguishes the different techniques for classification. There's usually an important trade-off involved here.

If the class of functions we look at is too simple, we won't be able to find the function in this class that does a good job describing the right way to map feature vectors to category labels. If we look at classes of functions that are too complicated we'll need immense amounts of data and possibly computation to figure out which one to use.

The first method we'll look at is called support vector machines, or SVMs. The category of functions they use will seem very simple, perhaps even too simple. But they're often very effective maps. Moreover, we'll see that there is a beautiful and simple technique called the Kernel Trick. That lets them handle a much more complex classes of functions with very little additional work.

In addition, they'll be the fundamental building block out of which we'll construct deep networks, which for many problems give the president state of the art. To specify SVMs we'll need to specify two things. The collection of allowable classification functions that we could end up producing. And how we choose which one to use based on the training data.

The classification functions we produced will be what we call linear classifiers. They'll work by applying a linear function and thresholding the result. More precisely, we will specify a classifier by giving a vector w of weights, w1through wn, where n is the number of coordinates in the future vectors.

And a threshold b, given a data point x, with coordinates x1 through xm. They first compute the dot product, w.x. Which you'll recall means that we add up w1 times x1, plus w2 times x2, etc., up to wn times xm. They'll then compare the resulting number to the threshold b.

If it's larger than b, they'll put the point in the positive category, otherwise they'll put it in the negative category. This has a nice geometric interpretation. If say the feature vectors are two dimensional, instead of points with w.x exactly equal to b, it's just the line in the plane.

The points with positive dot product lie on one side and the points with negative dot product lie on the other. The classifier is thus just cutting the plane into two pieces with a straight line. In three dimensions, the set of points with w.x equal to b is a two-dimensional plane, and ou classifier uses it to divide the three-dimensional space into two regions.

When the dimension is larger, it's a little harder to visualize, but the picture is analogous. The points with w.x = b form an n minus 1 dimensional hyperplane which divides un-dimensional space into two regions. The classifier calls something in one region positive, and something in the other region negative.

Now, let's see how we should choose such a hyperplane given a set of label points of training data. It's not always possible to separate the positive and negative points for the hyperplane. For instance, we can clearly see that there's no way to do it with the points showing here.

However, let's suppose for now that we're in the good case where we can find such a hyperplane that perfectly separates the positive and negative training points. In this case, we will call the points linearly separable, and it means that there is some linear classifier that correctly classifies all of the training data.

If you look at the picture, you'll see that there could be many such hyperplanes. While they all give the same answers on the training data, it would correspond to very different rules for classifying new objects. For instance, the two hyperplanes shown here both match the training data exactly.

However, they would give very different answers when asked to classify any of the points shown here. To choose among them, SVMs look for what's known as the hyperplane of maximum margin. Given linearly separable training data, we can see how far we can move this hyperplane until it stops classifying the data correctly.

In the picture, this means that we'll shift in one direction until it hits something in the set of positive points. And then we'll shift it in the other direction until it hits something in the set of negative points. This gives us two new hyperplanes, both parallel to our original one.

And they both correctly classify our training data. Intuitively, when these hyperplanes are very far apart, this means that the original hyperplane very robustly divides the two groups in the training data. Because it worked with room to spare so to speak, this means that it's reasonable to guess that if we had some new training data the hyperplane would probably still work fairly well.

We'll call the distance between these two parallel hyperplanes the margin. Now support vector machines work by looking at all the hyperplanes that divide the two groups in the training data. And then they try to find the one with the largest possible margin. If we imagine shifting and rotating the hyperplane around to see what possibilities are, you'll see that it can hit the points that lie nearest to it, shown here in red.

And that these are the ones that determine everything we need to know. Whereas the ones that are farther away, shown in blue, don't really matter. And moving them around a bit doesn't change the answer. The red points are called support vectors, which is why we call these support vector machines.

Here are a few examples of the maximum margin hyperplanes for different training sets. You can see that they nicely match our intuition about the right way to divide the two classes. Going forward, there are two very natural questions that we'll need to answer. The first is whether and how we can find this optimal hyperplane?

And the second is, what we should do when the points are not linearly separable? Answering these will be the primary goal of our next two sections.

3.10 Perceptron: Simple Classifier with Elegant Interpretation

In the previous section, we looked at how to frame binary classification problem in the setting of supervised learning. Let's briefly review where we left off, and define some notations to use going forward. We described each object to be classified as a list of d real numbers, which we call the feature vector.

We can think of these as points in d-dimensional space, which we'll call Rd. A binary classifier is then a function that maps each point in Rd to one of the two possible categories. If we label the positive examples with plus 1 and the negative examples with minus 1, we can thus succinctly describe the binary classifier as a function from Rd to the set containing plus and minus 1.

We're then given a bunch of training examples, which consist of points x1 through xn, in Rd, along with our correct labels, l1 through ln. We use these to decide what function f from Rd to plus or minus 1 to use as our classifier. Then, given a new point to classify, we get our answer by just applying the function f.

Deciding what class of functions to choose the classifier from is one of the crucial design choices to be made when setting up our classifier. And we decided to start with the case of linear classifiers, which choose a category by dividing space up into two categories using a hyperplane.

We can describe the hyperplane as the set of points with w .x equals b. Where w equals W1 through Wd is a vector in Rd and B as a number. If we use plus or minus 1 to label our categories, we can write or classify very cleanly. We want to return plus 1 is w .x is bigger than b, and minus 1 if it's smaller.

So, this just says that f of x is equal to the sign of w .x minus b. We then looked at the case where there was a hyperplane that correctly classifies all of our training data. In which case, we said that the points were linearly separable. We define the margin of such a hyperplane to be the amount we can shift in either direction, while still correctly classifying all the training data.

Given linearly separable training data, there'll often will be many hyperplanes that separate the positive and negative examples, so we have some freedom in how to construct our linear classifiers. One role people often use to make this decision is to choose a hyperplane with the largest possible margin. The resulting linear classifiers called the Support Vector Machine.

We thus gave a reasonable proposal for how to perform binary classification when the training examples are linear acceptable. But it's only useful if we can actually implement it reasonably efficiently. There are actually some problems that don't look too different from this. But for which the running time grows exponentially in the number of dimensions, which would be prohibitive, even for fairly moderate D.

So, it's not obvious apreari that we can do this. However, luckily, we can. Let's see how. Our classifier is specified by the vector w and the threshold b. For it to correctly classify the ith training example, we need w .x of i minus b to be greater than 0, if l sub i equals 1, and less than 0 if l sub i equals minus 1.

Keeping the vector b around ends up being a little bit annoying. It'd be cleaner if we were in the case where b equals 0. It turns out that there's a nice trick that lets us reduce to this case. And the idea is just that we're going to add the dummy coordinate and set it equal to 1 for all of our training points.

So, we now think of our training points as vectors, x sub 1 prime through x sub n prime, where x sub i is the d plus 1 dimensional vector X sub i comma one. We then pull b into our weight vector by defining w prime to be the vector w comma negative b.

This makes it so that w prime dot x sub i prime equals w dot x minus b. So, finding a vector w such that w dot x sub i minus b is greater than or less than 0 is the same as finding a vector w minus prime, such that w minus prime .x sub i minus prime is greater than, or less than 0.

And now we don't have to worry about b. We can thus assume from now on that b is 0 without loss of generality. With this pre-processing step, the algorithmic problem of finding a classifier that works for all of the training data comes down to finding a vector w that meets all the constraints coming from the x of i.

That is we want to find a vector w, such that w .x of i is greater than zero, for all i such that l sub i equals one. And w .x sub i is less than 0 for all i such that l sub i equals negative 1. Let s be the set of w that meet these constraints.

To find the support vector machine, we wanna find the w in s with maximum margin. If we start with the hyper plane w .x equals 0, we can get shifted versions of it by just changing the right hand side. How quickly the hyperplane moves when we change the right hand side depends on the norm of w.

And one can check if the hyper plane w dot x equals delta, corresponds to shifting the original one by a distance of delta over the norm of w. If we shift the hyperplane so that it hits a point xi, which corresponds to the margin of the hyperplane with respect to the point xi, this will be the hyperplane with delta equal to w .xi.

Which means that we shifted it by W. Xi over the norm of W. By scaling these constraints, we can write the problem of finding the hyperplane of maximum margin as just minimizing the norm of W. Now, there are a few ways one can find the W we're looking for.

In other sections of this course, both before and after this one, an important theme has been that we have these powerful algorithmic techniques available for solving convex optimization problems. In practice, the most common approach for finding linear classifiers in general, and FPNs in particular, is to use convex optimization.

The basic idea is that one can rewrite the problem of finding the hyperplane of maximum margin as optimizing a convex function over the set S. And we can then apply the convex optimization machinery. This machinery has been carefully tuned for SVMs, and there are excellent libraries available for doing this.

However, rather than just running through some algebra to write the problem in a convex form, and then referencing the existing general algorithms for these problems, I'd like to instead talk about a beautifully simple way of solving the problem, that I think actually might be a bit more insightful.

It's called the perceptron algorithm. And it's not quite as efficient as the convex optimization routines, and I'm just gonna talk about it for finding any linear classifier, rather than to find the one with with the largest margin. But it has some beautiful ideas in it. And it will let me give you a full, self contained algorithm.

Moreover, it solves a more general version of the problem called online learning. Before we talk about the algorithm, there's one final modification that it will be convenient to make. Since we've gotten rid of B, all of our constraints look at whether w .x sub i is positive or negative.

Note that scaling Xi by a positive content won't change that. For instance, saying that W .Xi is positive is the same as saying that W.5xi is positive. We can best scale all of the Xi however we want. By doing this, we can assume that each Xi has norm, which is just another word for length equal to one.

A basic property of dot products says that the square of the norm of a vector is given by the dot product of the vector with itself. So, we can write this as norm of x squared equals xi .xi equals 1. We can now finally describe the output. The idea is as follows.

Instead of being given all of the training data, we'll imagine that somebody hands us the points from it one at a time. Possibly giving us the same point more than once. Each time we're given a point, we have to assign it to a category. After we do, we find out what the actual label was, and we can use this information for the later points.

Now, the difference here is that we have to make our decisions online. That is, we don't get to see the whole training stat before we have to classify some points. Our algorithm for doing this is going to be very simple. We'll maintain a vector w that will be our guess for the weights, which is initially set to zero.

Given a point, X, to classify, we'll then use your current guess for W. So, we'll guess positive if W .X is greater than 0, and negative otherwise. Now, if we guessed right, we'll leave our guess for W unchanged. However, if you make a mistake, we'll update our guess to try to fix it.

Specifically, if we guess negative, and the answer was positive, we'll replace w with w plus x. This increases w .x, so it pushes w in the direction of answering positive for x. Similarly, if we guess positive when we should have said negative, we'll replace w with w minus x.

Now, if we wanna solve the original problem, we can just pick our points from the training set, say randomly, and feed them to the algorithm and repeat. That's actually the whole algorithm, which I've written a bit more formally here. We can actually show a remarkably strong guarantee about this algorithm.

It will say that if there is any linear classifier that correctly classifies all of the points that were given with some margin gamma, then the total number of mistakes we'll make over the entire series of points will just be one over gamma squared. To give you a feel for it, here's an animation of the algorithm being run on some two dimensional data.

Here are the data points, and here is the maximum margin hyper blade, which we notice does not pass through the origin. We start by adding a dummy coordinate to let us work with hyper planes through the origin, as shown here. And by initializing w1 to be 0, we'll choose each x sub t randomly from the data points, allowing reads.

Now, one mild technicality is that w1 is 0. So, for the first point, the dot product, w sub 1.x sub 1 is actually 0. And we need to decide whether to call that. Positive or negative. Which one we choose doesn't really matter, so we'll just pick one. Say, negative, if the dot product is exactly zero.

Now, in this case, the correct label for x1 was positive, so we made a mistake. And we update our weights by setting w2 equal to 0 plus x1. We now have a nonzero weight vector, so we can think of it as specifying a hyperplane, which we then use to classify the points.

We'll then repeatedly compute W sub .Xt, use that to see which side of the hyperplane the point lies on, and then use this to compute the sign, which we'll use to compute a prediction for l sub t. We'll then use this to update our weight vector accordingly, and obtain W sub t plus 1.

The rest of this section is an animation of the perception algorithm run on our data. It steps through the first ten iterations slowly, so you can see exactly what's happened. And then it quickly steps through the next hundred, so that you can get a feel for the overall behavior of the algorithm.

For the first ten iterations, you may find it helpful to pause the video on each and make sure you understand how I got from one step to the next.

3.11 Perceptron Proof

In this section, we'll prove our previous claim that the perceptron algorithm makes at most one over gamma squared total mistakes, where gamma is the margin. I should mention that this section may be slightly more mathematically involved than the others. I think it provides a lot of insight into why the perceptron algorithm works, so I decided to include it.

But nothing we do later will rely on the details of this proof. So if you skip this or you don't fully follow the details, it won't cause any problems with the later material. Okay, so let's now see the proof. Our starting assumption is that there's some linear classifier that classifies all of the x sub i correctly with margin gamma.

With some gamma greater than zero. We can describe this classifier by giving its weight vector w star, which we'll normalize to have norm one. Because of this normalization, the distance of a point x sub i from the hyperplane is just given by the absolute value of w star dot x sub i.

The margin assumption says that w star dot x sub i is greater than or equal to gamma whenever l sub i is one and w star dot x sub i is less than or equal to negative gamma whenever l sub i is negative one. Now, let w sub tb the way vector after the algorithm has gone on for t iterations.

The idea of the proof is that we're gonna keep track of the dot product w sub t dot w star which corresponds geometrically to the component of w sub t in the direction of w star. If w sub t perp is the component of w sub t that is perpendicular to w star, as shown here, then the length of w is given by the Pythagorean theorem.

Which tells us that the norm of w sub t squared equals the sums of the squares of these two components. So the norm of w sub t squared = (w sub t dot w star) squared + norm w sub t perp squared. Since the norm of w sub t perp squared is always at least 0, this tells us in particular that the norm of w sub t squared is always at least w sub t dot w star squared.

So the norm of w sub t is always at least w sub t dot w star. Geometrically, this is just saying that the length of the whole vector, w sub t is at least the length of the component in the direction of w star. Now the idea behind our analysis is to show that w sub t dot w star will go up a lot every time we make a mistake.

If we then show that the total length of w sub t doesn't go up too much overall this will mean that we couldn't have made too many mistakes. Let's make this precise. In step t we see the point x sub t and predict the label by looking at the sign of w sub t dot x sub t.

If our prediction is correct, we don't change our weight vector. So, w sub t plus one equals w sub t. And the dot product of our weight vector with w star doesn't change. The interesting case is when we make a mistake. Our first claim is that this will substantially increase the dot product with w star by making w sub t plus 1 dot w star greater than or equal to w sub t plus gamma.

To show this, let's first suppose that we have a false negative in step t. This means that we predict negative, but l sub t equals positive 1. So we then update our weights by setting w sub t plus one to equal w sub t plus x sub t.

So w sub t+1. w star equals w sub t. w star which equals, w sub t. w star + x sub t w star. Our margin assumption has that x of i. w star is greater or equal to gamma. Whenever l sub i is 1. So the right had side is at least w sub t dot w star plus gamma.

Geometrically, this says that the assumption that w star correctly classifies the point of margin gamma, means that x sub t has to have a component of at least gamma in the direction of w star. So adding x sub t increases the component in the w star direction by at least gamma.

The case of a false positive is similar, just with some signs flipped. A false positive means that we predict positive when l sub i = -1. And we update our weights by setting w sub t +1 = w sub t- x sub t. So we get that w sub t + 1 dot w-star = (w sub t- x sub t) dot w-star.

Our margin assumption says that x sub i dot w-star is less than or equal to negative gamma whenever l sub i = -1. So we get that the right side is at least w sub t, dot w star, minus negative gamma. Which again equals w sub t plus gamma.

So in both cases, we got that the dot product, increases by at least gamma, as we claimed. Now our second claim is that the total length of the weight vector, doesn't increase too much overall. Again, nothing changes when we classify the point correctly, so we really only have to worry about the case where we make a mistake.

In this case, we'll show that when we make a mistake, the squared length of the wave factor never increases by more than 1. That is, that the norm squared of w sub t plus 1 is less than or equal to the norm squared of w sub t plus 1.

Showing this just takes a few lines of algebra. Let's first do the case of a false negative. In this case, we classify x sub t as negative, so w sub t is less than or equal to 0. And we set w sub t +1 to equal w sub t +x sub t.

The norm squared of the vector is just the dot product of the vector with itself. So the norm of w sub t +1 squared = w sub t+1 dot w sub t + 1. Which we can expand out as w sub t dot w sub t + x sub t dot x sub t + 2w sub t dot x sub t.

The first term is just the norm squared of w sub t and the second is just the norm squared of x sub t. So this equals the norm squared of w sub t + the norm squared of x sub t + the last term, 2 w sub t dot x sub t.

Now, remember that we normalized x sub t to have a norm 1. So the norm of x sub t squared just equals 1. And w sub t dot x sub t is negative. So we gather the norm of w sub t plus 1 squared equals the norm of w sub t squared plus 1 plus something that's less than or equal to 0.

So it's at most the norm of w sub t squared plus 1. We can interpret this geometrically as well. The fact that we classify x sub t as negative means that the dot product between x sub t and w sub t is less than or equal to 0.

Which means that the angle between these two vectors is at least 90 degrees. And x sub t has length 1. If the dot product is 0, the angle's exactly 90 degree. And the norm of w sub t + x sub t squared is exactly equal to the norm of w sub t squared + 1, by the Pythagorean theorem.

And this is the worst case, since the vector only gets shorter if the angle is more than 90 degrees. The case of a false positive is again the same, with some signs flipped. This time, w sub t dot x sub t is positive, and the norm of w sub t + 1 squared = (w sub t- x sub t) dotted with itself.

Which expands out to the norm of w sub t squared + 1- 2 times the positive number w sub t dot x sub t. Which again, is at most the norm of w sub t squared +1 as we want it. Now, we can put these two claims together to bound the total number of mistakes made by the algorithm.

Suppose that we run our algorithm for say m steps and we make a total of k errors along the way. At the beginning of the algorithm, we start with a weight of 0. So the initial dot product, w sub 1 dot w star equals 0. Claim one tells us that the dot product increases by at least gamma every time we make a mistake.

So if we make k mistakes, we know that the product at the end, w sub m dot w star, is at least k times gamma. On the other hand, claim two tells us the norm of w sub m squared. Goes up by, at most, 1, each time you make a mistake.

So the norm of w sub m squared is, at most, k. And therefore, the norm of w sub m, is, at most, the square root of k. Now, as we said earlier, since w star has norm 1, the dot product, w sub m.w star is the length of w sub n's component in the direction of w star.

And this can't be bigger than the length of the whole vector w sub m. So, w sub m dot w star is less than or equal to the norm of w sub m. Putting these all together, the yet that k gamma is less than or equal to w sub m dot w star which is less than or equal to the norm of w sub m which is less than or equal to the square root of k.

So k gamma is less than or equal to the square root of k, which means that k is at most 1/gamma squared, which is what we originally claimed.

3.12 Perceptron & Data That is Not Linearly Separable

We've seen that we can use support vector machines for binary classification, as long as our training data is linearly separable. However, as you might guess, not all data sets are linearly separable. And we'd really substantially limit the applicability of our techniques if these were the only classification problems that we could handle.

For instance, we could get data from some experiments and these experiments maybe should lead to nice linearly separable training data. But possibly they have experimental errors that lead to a few errant data points in our training data. Or we could be in a setting where there's a good classifier but is just isn't the hyperplane.

In this section, we'll give a brief overview of some of the simple modifications that allow us to apply support vector machines in these more complicated settings. In the first case, where we want a linear classifier, but there's no hyperplane that perfectly classifies the training data, the idea is just to slightly modify our goal.

Instead of requiring our hyperplane to cross by all of our points in some given margin, we'll make a penalty function that zero for a point that is classified by the hyperplane with this margin and that penalizes points that violate the constraint. And we'll make the size of the penalty depend on how bad the violation is.

Then given some margin, we can look for the hyperplane that has the smallest penalty. Note that there's a tradeoff here. If we demand a larger margin, we should expect more violations and thus a larger penalty so we have to somehow decide how to balance the two. There's also some discretion in how we choose our penalty function.

Researchers have studied how to make these choices, and there's some simple, commonly used penalty functions that work very well in practice. And in fact, for these penalty functions, it turns out we can actually find the optimal hyperplane using algorithms that are almost the same as the ones we used in maximum margin setup.

Let's now talk about the second problem I mentioned which actually seems to be a bit more problematic. This is the problem where there is a right classifier, but it's just fundamentally not a hyperplane. For instance, it could be that the best way to classify the data is by looking at some higher degree polynomial instead of a linear function.

To see how this could happen, let's look more carefully at the example I showed before. In this case, any linear separator that we try will miss-classify a significant faction of the trait data. Here's a picture of what this would look like. However, we would correctly classify all the training data and get what visually seems to be the right classifier by looking at the degree two polynomial x squared + y squared, labeling a point as positive when it's bigger than one and negative when it's smaller than one.

Unfortunately, our whole approach was tuned into finding a hyper plan so seems like we need to start from the scratch and try with all of the theory and all the algorithm that we have. Before completely anew if we wanna get to classifier that has some other shape. Like the ones we got to be higher.

Our priority we don't even know if we can do this on a reasonable way. Our algorithms, like the perceptron algorithm we described, or the convex programming methods that I mentioned, were all really about linear separators. Priori, it actually seems quite possible that finding the best classifiers with say degree two polynomials could just be fundamentally harder, and end up being too hard a problem, one that we can't solve with any efficient algorithm.

However, as you may have guessed, this isn't the case. There is a beautiful trick that actually lets up use what we know about support machines directly to find a whole bunch of different types of non linear classifiers without having to start from scratch. The basic idea is really simple.

Remember that when we set up the problem, we have to choose how to encode our data as feature vectors. But this was a design decision, and there are a lot of ways we could've made this choice. For instance, instead of making the coordinates of the future vector be x and y, I could have taken them to be out of 100x squared and 74cos y, or x + y and xy.

Or I could have even put them in three dimensions, with coordinates say xy and x cubed times y. In general, we can apply any function that we want to our data points, and use the results as our feature vector instead. Now, how we make this choice has a huge effect on what a linear classifier looks like, what its margin is, and how well it performs and different choices can actually lead to very different answers.

In fact, we already took advantage of this freedom a little bit when we were describing the perceptron algorithm. Right at the beginning of the algorithm we decided that it was easier to only work with hyperplanes through the origin. The only way were able to get away with this was by adjoining a dummy coordinate of one.

That is by applying a function that takes a vector x to a new vector x,one in a space that's one dimension larger. This then broadens the types of classifiers we could get by taking hyperplanes through the origin. Any classifier we could have gotten in the original setup by using some arbitrary hyperplane that might not have gone through the origin, can now be expressed as a hyperplane through the origin in this higher dimensional space.

One thing to note here is that by lifting the points to a higher dimensional space, we actually gave ourselves the ability to describe more classifiers as hyperplanes through the origin. Intuitively, we can describe a hyper plane through the origin in d dimensions, by giving a weight vector with d coordinates.

By throwing in a dummy coordinate, we put our vectors into d plus one dimensions. So we now describe a hyper plane with d + 1 numbers. So, by lifting our points up into a space, with one more dimension, we gave ourselves one more degree of freedom, to use in describing our hyper plane.

But that just seems like a little technical convenience to avoid working with numbers other than 0 on the right hand side of our equations. The idea of the kernel trick is to really exploit this freedom. If we wanna look for classifiers whose values are delineated by something other than a hyperplane, we'll apply a function f that maps our original points to some new points, so that these more complicated classifiers on original points are just linear classifiers applied to the new one.

To see how this could work, lets go back to the example we had before. If we had somehow encoded our points differently, say like this. Then we could have gotten a much better result with a linear classifier, as shown. Now this is pretty adhoc, but there's actually a more systematic way that we can proceed.

Instead of describing a point with two numbers, X and Y Let's describe it as six numbers. One, X, Y, X squared, Y squared, and XY. Formally, we can think of this as a function that takes X, Y to a new vector with six coordinates. Let's call them Z one through Z six.

By applying the function f written here. This didn't really add any information. We're still just giving a more complicated way of encoding the original numbers X and Y. But now we have six degrees of freedom for our hyper planes through the origin which we can describe by saying the linear function c1z1 plus c2z2 up to c6z6 equal 0.

But if we applied this linear function to f of xy for one of our original points we get c1 plus c2x plus c3y plus c4x2 plus c5y2 plus c6xy. But this is the general form of a polynomial of degree at most two applied to x comma y. So, by choosing c one through c six, we can describe and quadratic polynomial.

And the linear classifiers applied to z give quadratic classifiers applied to x comma y In particular if we let c1 = c4 = c6 = 1 and then set the rest to zero, we get the classifier that returns positives if 1+z4+z5 which equals 1+x squared+ y squared is positive and then negative otherwise, which was exactly the one that looked like the right choice for our sample.

This is actually kind of remarkable if we want quadratic classifiers, we don't need to start over. We can get them by just applying a function that lift our points up into some higher dimensional space, and then look for linear classifiers. We actually could've done this more generally. If we wanted degree functions, we just need to include the degree three.

X cubed, XY squared, X squared Y, and Y cubed. In general, if we want degree K classifiers, we just need to apply the function that writes a point by listing all the monomials of degree at most K. We can then just apply our SBM algorithm to the results.

So we don't need to do any additional thinking grees. All we have to do is just apply a function and use the SVM machinery that we already have. And there was nothing that forced us to use polynomials. If we wanted to describe some other sort of classifiers, we could have just applied some other function to our training data instead.

Now you might worry that the computational problems is getting a lot harder, since were getting a lot more coordinates. For example they are on the order of d to the k mono nomials of degree k in d variables, so this gets big in a hurry. However, it turns out that we don't always need to explicitly work with these really high conventional representations.

If we look carefully at the perceptron algorithm or at a lot of the other algorithms we're finding FEM's. It turned out that we can actually describe in a way that doesn't specifically need to know the coordinates of the data themselves. Instead it just needs to be able to compute dot products, f(xi) and f(xj).

So if we want to find linear separators after applying the function f, we just need to be able to compute dot products of the forum f of x sub i dot f of x sub j. Often, there's a way to do this that doesn't require us to actually compute all the coordinates.

In particular, for the function f that takes a point to all monomials of degree up to k If we scale the coordinates by some appropriate constants, the dot product of f(x sub i) and f(x sub j) turns out to just be given by (1 + x dot y) raised to the dth power.

To find this we just need to compute the dot product x dot y. We never actually need to write out this long list of variables. This is obviously a lot faster, and it lets us handle cases that would be way too complicated otherwise. So, to abstract a little and sum everything up, the idea was that we would get more complicated classifiers by applying a function F to our feature vectors, and then using SVMS on the results.

Instead of computing F of x for all points, we actually just need to compute the dot products, K of x of i comma x of j, which we'll say equals F of x of i dot and F of x of J. And we call this the kernel trick.

This kernel is all we actually need to be able to run our algorithms. This idea is called the kernel trick and it lets us apply our SVM machinery to get non linear classifiers. This is an extremely powerful tool. But I just like to conclude with one. One quick caveat, remember that when you have more parameters, you need more data to properly fit them, otherwise there's a risk of so called overfitting where you get a really complicated function to describe your data, but this function is fairly useless on new data points as shown here.

When using these kernels, it actually gets easier to do this since we're introducing a lot more possibilities for our classifiers. One thus has to use some care in applying them and they have to be used judiciously to avoid overfitting. There's a beautiful theory that studies the tradeoffs in doing this.

But it's slightly beyond the scope of this course. In general, you should think of the goal as trying to work with as few degrees of freedom as you can, while still being able to describe a classifier.

3.13 Estimating the Parameters of Logistic Regression

It is now time to discuss methods for estimating the parameters of logistic regression. Namely the parameters such as beta 0, beta, beta w, and beta g. As we noted earlier, these parameters are estimated using historical data. In particular, the data will be used in combination with a so called maximum likelihood estimation.

Let us first describe the general principle in abstract terms and then illustrate it with a simple example. Suppose, we have a collection of the data in the following form. Here X1, X2 and all the way to Xn are observations for independent variables. And Y1, Y2 and all the way to Yn indicate whether the event of interest takes place.

In particular, Ys only take values 1 and 0. Here, the convention is that Yi takes value 1, if the event took place for the ith element of the sample, and takes value 0, otherwise. Now, remember, that according to the logistic regression model with one independent variable, we have the following relationship between the observations and data.

Say, for example, that Y1 equals 1, then we have the following relationship. On the other hand, say, for example, that Y3 equals 0. Then according to our model, we have the following. So, that we don't have to indicate what case we are dealing with, observe that we can combine both cases as follows.

Please take some time to understand the meaning of this expression and convince yourself that the trick works regardless of whether the value of Y equals 1 or equals 0. One crucial assumption is that the observations are independent, just like it is a common assumption in the linear regression case.

This is often the case in practice. Say, once again, X corresponds to individual's weight and Y indicates whether the patient has diabetes. It stands to reason that these observations are independent. If this is the case, the probability of observing a particular outcome for all individuals from 1 to n simultaneously is the product of probabilities for individual outcomes.

And can be represented by the following expression. Now recall that the values of Xs and Ys are actually known to us. These are what we have collected in our data sample. The only unknowns are beta 0 and beta coefficients, which we want to estimate. The maximum likelihood estimation is a principle which suggests that the best estimation of beta0 and beta are those which makes the expression on the right hand side of the equation as large as possible.

In words, one could say, the best estimate for beta 0 and beta are those which make the likelihood of this particular outcome as large as possible. If this was too abstract, no worries. Let us turn to a concrete example to illustrate this idea. Imagine, that we have the following data on some individuals.

Their body weight and whether or not they have developed diabetes. For simplicity, we will consider the case of a small data size, consisting of only ten data points. But naturally, in practice, you would want to use methods with a larger number of data samples for better accuracy. The left column simply gives the list of body weights for the ten people.

The right column indicates whether the corresponding individual has diabetes. The value 1 stands for the case when the answer is yes, does have diabetes. And the value 0 stands for the case when the answer is no, does not have diabetes. For example, the individual with weight 186 pounds corresponding to the first row in the table does not have diabetes.

And the individual with a weight 156 pounds corresponding to the third row of the table does have diabetes. Then according to our formula, we need to find beta 0 and beta, which maximize the following expression. Take some time to study this expression, and convince yourself that we have applied the formula correctly.

Now ignoring the term where we multiply by 0s, we can simplify the formula as follows. So, we need to find beta 0 and beta which make this expression as large as possible. How can we do this? Well, one straightforward approach is to try all possible values of beta 0 and beta within some range, by say, iterating over small increments.

For example, try all beta 0 and beta between -10 and 10 with increments say 0.01 and select the pair which gives the largest value. This method however, has several important limitations. Number one, we cannot be sure that our range between -10 and 10 includes optimal values. Number two, our increments can be too crude.

Select and say, increment of 0.01 for beta 0, might be too refined and at the same time too crude for the second coefficient beta. Finally, when we have more than one independent variable, recall that in our second example, we had two independent variables, body weight and the number of parents with diabetes.

The time to try all values within their fixed range might be too large for practical purposes. So, how do we find the best values? This brings us to the third and last part of this lecture. Algorithms for estimating regression coefficients. There is a short answer to the task of finding the best regression coefficients, beta 0 and beta.

Simply use one of the standard statistical packages. Most of them have the capability to do so. In fact, I would recommend to use one of them until you become a sophisticated practitioner of statistics. But we would like to leave the lead, so to speak, and share with you the main ideas for the method.

This method will be discussed in some of the later lectures, so it is worth discussing it now as well. The secret is that the expression we'll need to maximize in terms of beta 0 and beta is actually very nice if we instead take the logarithm of this expression, and as a result, get the following formula.

It turns out that this new function of beta 0 and beta, even though it involves a complicated combination of logarithms and exponents, falls into a very nice class of functions called concave functions. Let us imagine we have the function of just one variable. Now the expression we need to maximize has two variables, namely beta 0 and beta.

But let us imagine for now that we're dealing with the function of only one variable. Such a function is called concave, if it has the following shape. The definition of a concave function is that whenever we draw a horizontal line between any two points on the curve, the line stays entirely below the curve, like this.

Or like this. The concave function is particularly nice when it comes to finding its largest value, namely solving the maximization problem. There is a variety of methods, including the so called Newton's method or the gradient descent method, which find the maximum values for such functions. These methods work by finding larger and larger values of function by small incremental improvements, sort of like what you see on this picture illustrated by small arrows.

Until the value of x corresponding to the largest value of the function f of x is found, like this. The crucial concavity property guarantees that these steps will converge to the point which is the maximum point for the function f. There is a very similar nice class of functions called convex functions, which look like this.

And which are good for finding a minimum point on the function instead. Let us imagine, we have access to the statistical software, which applies one of these methods, and indeed finds the values of beta 0 and beta which maximize the expression of interest here. I have done it myself, and found that the best values after some rounding are as follows.

To summarize, according to this model, the logistic regression model prediction of the impact of body weight on the likelihood of developing the diabetes is given by the following formula. Naturally, the numerical values for beta 0, and beta, we have derived above, depend on the data we have used for the estimation.

When the data changes, the estimation values for the parameters change as well. There are many additional questions worth discussing regarding logistic regression model, such as confidence intervals for the regression coefficient, statistical significance, relevant ranges, dealing with outliers for which we simply do not have time in this course.

For now though, let us turn to a very important real life application of the method to the reliability analysis of O-rings for Challenger space shuttle. Was there enough evidence to convince NASA managers that the launch was too risky?

3.14 Case Study: Challenger Launch

Now, we turn our attention to the Space Shuttle Challenger disaster that occurred in January 28, 1986. Through this case study, we would like to highlight how the application of the tools you have already learned, specifically logistic regression, could have allowed the management of NASA to potentially arrive at the different conclusion when deciding whether to postpone the launch of the shuttle on that fateful day.

We should note that some of the details, data, and circumstances are different from those that actually took place and were simplified for the discussion purposes. The high level ideas are the same, though. Some background. Up until January 28, 1986, the Challenger had been launched 24 times with no catastrophes.

A catastrophe can occur, in particular, if the collective failure of all these so-called O-rings that we have discussed at the beginning occurs. However, 7 of these 24 launches had experienced at least one O-ring failure. During one of such launches, in fact, three of five rings failed. One launch had two O-rings failed, and five launches had a single failure.

So collectively, there were ten O-ring failures. On the day of the 28th launch, which had already been postponed a few times due to bad weather and other technical difficulties, the temperature at the launch site, Kennedy Space Center, Florida, was unusually cold with morning temperature close to 30 degrees Fahrenheit minus one degree Celsius.

Some engineers expressed concerns about the launch in such cold weather given that there was no data to certify this successful launch at those temperatures. It was eventually decided to go ahead with the launch. Tragically, Challenger experienced a catastrophe were all of the five O-Rings failed leading to the destruction of the shuttle and killing everybody on board.

The launch and the subsequent disaster was broadcasted live on television to a large audience across the United States, making headline news. President Reagan made a special announcement in place of the original scheduled State of the Union Address. When reviewing data from the previous 24 launches, the team at NASA, which notably did not include a statistician, decided to look at the temperature at the time of launch for all O-ring failures.

Effectively, they looked at the plot shown here. This figure is the scatter plot for the temperatures at launch and the number of O-rings that failed. There were two launches with one failure, which occurred at temperature 70 degrees Fahrenheit, which is indicated by a bigger blue circle. Notably, this is data only for the seven launches when the failures of O-rings did occur.

This plot appears to hint that there is no significant relationship between the temperatures and O-ring failures given that the failures have not exclusively occurred at lower temperatures. However, as we said, this analysis ignores valuable information available from the 18 other launches where no failures occurred. A complete plot looks like this.

The marks in red show launches where no O-ring failures were detected while those in blue represent one or more O-rings failures as before. The picture now appears quite different, doesn't it? It is clear that higher temperature tend to result in safer launches while there is a skew towards lower temperatures of O-ring failures.

Alas, the absence of this more complete analysis could have provided deeper insight to inform NASA's decision as to weather to go ahead with the launch on the morning of January 28th. In fact, in the aftermath of the disaster, one of the recommendations of the investigation commission was to have a statistician be a part of the control team.

Savvy students such as yourself would not accept a scatter plot as confirmation of the relationship between the O-ring failures and temperatures. Indeed, while the scatter plot visually depicts a relationship, we now have a tool to be more rigorous than that. Since this problem involves a binary classification, that is failure, no failure, and we have an independent variable, that is temperature at time of the launch.

It appears that logistic regression might come in handy to establish a relationship between independent and dependent variables and crucially help predict the probability of failure given the core temperature of the morning of January 28, 1986. Now, let's see how we might build such a model. First, let us look at the following table, which gives the number of failed O-rings for each of the 24 launches along with temperature during the launch.

First, note that for each of the 24 past launches, we have 5 independent data points since the shuttle has 5 O-rings. For example, two launches took place at 70 degrees Fahrenheit with no O-ring failures during either of the launches. So we have collectively 10 data points, 5 for each launch corresponding to this temperature, which is classified as no failures.

All in all, we have 24 * 5, that is 120 data points from past launches. For our dependent variable Y, we have the binary states. No failure or failure, which can be represented as zero or one, where zero corresponds to no failure and one corresponds to failure. So out of these 120 data points, we have ten failures, value 1 in column Y and 110 no failures, value zero in column Y.

For the independent variable X, this problem only has the temperature at the time of the launch in degrees Fahrenheit. Our data is shown here. We are interested in the likelihood of O-ring failure. That is Y equal to one, given the lowest temperature on January 28, 1986. The temperature on that morning was 36 degrees Fahrenheit.

Specifically, we are interested in the following likelihood. The probability that Y equals 1 conditional on the temperature X equals 36. According to our logistic regression model, this is expressed as follows. The probability that Y equals 1 given that X equals 36 is as follows. In order to determine this probability, we need to use logistic regression to determine the coefficients beta zero and beta using the 120 data points from past launches.

Using logistic regression library in your favorite environment, R, Python, etc., using the 120 data points, we'll get the following maximum likelihood values for the coefficients. The coefficients we are most interested in is beta, because it helps us determine the relationship between the temperature at launch and the failure of O-rings.

At the 95% significance level, the confidence interval for beta is from -0.5 to -0.2, which does not include zero, implying that the coefficient for beta is significant. Alternatively, the P-value for this coefficient is 0.002, which is less than 0.05 and confers the same information about the significance of temperature in determining the no failure versus failure of O-rings.

This is already something. We have determined with 95% confidence that temperature does impact the likelihood of O-ring failure. The negative sign for the coefficient beta tells us that the temperature in O-ring failures are inversely related as we have intuitively determined earlier. Specifically, all things being equal, the odds of failure decrease by roughly 14% when the temperature increases by one degree Fahrenheit.

We can find this from the following derivation. Similarly, the odds of failure increase by roughly 16% when the temperature decreases by one degree Fahrenheit. Note that these are the proportional changes of odds and not the actual probabilities. We now turn our attention to answering the questions we set out for ourselves.

What is the probability of failure for each of the 5 rings on the morning of January 28, 1986 given that the temperature at launch was 36 degrees Fahrenheit? We'll use our logistical regression model to answer this question. First, the probability that one can create O-ring fails, let's say the O-ring labeled OA is found to be as follows.

The probability that Y is equal 1 conditional that x is equal to 36 is as follows. There is almost 90%. This is quite significant probability of failure. But does that also answer our question about the probability of catastrophe? Recall that there are five O-rings, and we'll have to determine the probability of failure for each of them individually in order for us to determine the probability of a catastrophe.

We need to determine the probability of failure for all five such O-rings. The failures of O-rings at a given temperature are independent events, thus the probability of a catastrophe is the probability of one of the O-ring failures multiplied by itself five times. Namely, it is 0.89 raised to the power 5, like this.

According to our logistic regression model, the probability of a catastrophe on that fateful morning was as high as 57%. Had such an analysis has been performed before approving the launch, it is quite unlikely the decision to launch would have been taken by the management of NASA. For reference, the table below shows the probability of failures for individual O-rings in the corresponding probability of a catastrophe at various temperatures according to our model.

Caution. While it may seem like logistical regression alone could have prevented the Challenger disaster from happening, we must use caution in our conclusions. There are several reasons why the logistic regression model presented here must be accepted with some caution. The model we used is very simple. It only incorporates a single independent variable, temperature, at the time of launch.

For a system as complex as the Space Shuttle Challenger, it is hard to believe that the management and the engineering teams would be happier relying on such a single variable logistic regression model to predict the probability of failure of a very complex engineering system. To get the reliable likelihood of failure, one would need to take into account as much complexity of the real system as possible.

But this arguably simple model would still raise enough flags to stop the launch and perhaps to do a far more rigorous analysis. Notice that while we insisted on using all of the data available to the teams at NASA, there was not a single data point that included a launch temperature close to the 36 degree Fahrenheit observed on January 28, 1986.

As such, the model is extrapolating the relationship based on available data to a region, which is significantly outside of the available data. We can counter this by suggesting to use a 55 degree temperature as a benchmark as opposed to 36 degree temperature. After all, launches did take place at temperatures lower than 55 degrees.

The likelihood of a single O-ring failure, according to our table, is 0.34 at this temperature and the likelihood of a catastrophe is 0.34 raised to the power five, which is about half a percent. Would you proceed with the launch with such odds of failure, especially given that the actual temperature was much colder?

In summary, while our estimate of 57% likelihood of a catastrophe can be rightly criticized, there was certainly enough evidence that the likelihood of the disaster was significantly high and the launch should have been postponed.

3.15 Misapplications of Statistical Techniques

For some other subtle ways that misapplication of statistical techniques can lead to an incorrect conclusion lets turn to an example in the setting of scientific research. If you read the paper or browse the Internet, there's no shortage of advice, supposedly backed by some scientific study about the health effects of some food or behavior.

You could find say, red wine prevents heart disease, or antioxidants make you live longer, cigarettes cause cancer, coffee causes cancer, coffee prevents cancer, or maybe, some obscure berry causes weight loss. Some of these assertions like the link between cigarettes and cancer, stand the test of time, get confirmed by other scientific experiments and become acknowledged as true scientific facts.

For others, new studies don't find the same effect or maybe they even find the opposite one. And a literature goes back and forth with seemingly conflictory claims. This isn't just a problem with nutrition literature, you can find similar examples in the literature discussing the best treatment for medical problem, the genetic basis for disease, or the existence of telepathy.

You can find such conflicting evidence, even in clinical trials published in top scientific journals. Researchers who have studied this phenomenon have found surprisingly high rates of nonreplicability for scientific results. And some have even proposed the possibility that in certain fields, a majority of the published research findings could actually be false.

This should be pretty surprising, since these studies are rigorously refereed. And they're usually accompanied by p-values and confidence intervals and tables of data that provide a veneer of the epistemological comfort conveyed by modern science. What I'd like to do now is give a few examples of the commonly made mistakes that can cause these problems.

Consider the following scenario. Suppose we get frustrated with all the conflicting claims that we're reading. And we want to decide once and for all which foods cause or prevent heart disease. To do this, we collect data on the 100 most common foods that people eat and their incidence of heart disease.

Of course, designing this study is complicated. People lie about their eating habits. It's not always clear whether someone does or doesn't have heart disease. People eat foods in different quantities, etc. For simplicity, let's just set this aside and assume that we're able to successfully assign p values to hypotheses of the form, food x causes heart disease.

We then take all the foods that have p greater than 95% and we'll report them as causing heart disease with 95% confidence. Now, here's the problem. Say that something has a p value of 95%

means that under the null hypothesis that there's no effect, the chance of the observed outcome is at most 5%.

Let's suppose that none of the foods that you test actually have any effect on heart disease. This means that for each of the hundred foods, the null hypothesis that the food does not cause heart disease holds. However, if you get false positives for each with a probability of 5%, you'd actually expect to find an apparent effect for five of the hundred foods.

Even worse, if the false positives for different foods are independent, the probability that at least one food appears to have an effect is actually very high. It turns out to be about 99.4%. So even though the 95% constant seem quite good, the chance of you reporting a spurious effect is extremely high.

The problem here is what you might call hypothesis shopping. Instead of fixing a hypothesis and testing it, we looked at our data and tried to find a hypothesis that would pass the test. This probabilistic phenomenon, where we're simultaneously testing a large number of hypotheses, can occur fairly easily under less contrived conditions.

For instance, imagine testing each of five variants of a drug on five different types of cancer and then looking at how the effects breakdown by age, gender and ethnicity. It's easy to find even more extreme examples in scientific settings. For example in genomics, scientists often want to figure out what genes cause the disease.

A way to start looking for these is with what's known as a genome-wide association study, where they cross-reference the incidence of the disease with tens of thousands of genes, looking for the handful of genes that cause it. If they call some correlation significant and it clears the 95% confidence bar, you'd expect thousands of false positives, that is genes that exhibit a seemingly statistically significant correlation but don't actually have anything to do with disease.

Compared to just a handful of true positives, corresponding to the actual genes they are looking for. Of course, good researchers carefully correct for this, by for example, requiring much higher probabilities or just using these studies to identify candidates and then carefully testing them by other means. But it's easy to miss this and to make mistakes.

Alternatively, suppose you set up an observational experiment where you set something up and just look to see if anything interesting happens. There are a lot of potentially interesting behaviors. So you might actually expect something interesting to happen just by chance. This last one actually leads to a very subtle way that these biases can creep into the literature.

In general, it's pretty hard to get a good journal to accept a paper that says I did the following experiment and nothing interesting happened. People are much more likely to publish papers that describe some new and exciting phenomena. If one uses 95% as the cut off for something for being significant, you'd get something significant by chance 5% of the time.

But people are often much more likely to report their findings when they see something significant. So if people don't properly correct for this, you'd expect a lot more than 5% of the interesting phenomena that are submitted for publication to have occurred by chance. This is particularly acute if you have a bunch of groups of people studying the same phenomenon.

For instance, suppose that some food has no effect on whether you get cancer or not in actuality, but people think that there are some reason that it might. Suppose that this results in 50 groups of people independently performing experiments to test this hypothesis. Of these, you wouldl expect that few groups seeing an effect that clears the 95% confidence bar, maybe two or three.

These groups don't know about the others, so they excitedly submit their work for publication. The other 40 something groups don't see anything interesting and many of them don't even bother do publish this sort of finding. This means that all anybody ever sees is that two or three groups independently observe the phenomenon with 95% confidence, which if they were the only ones looking at the question, seems extremely unlikely to happen by chance.

This isn't even a mistake on their part, if they don't know about the other groups, they might not have any obvious reason to worry about this sort of problem. One reason that we're mentioning these sorts of issues is that they can become particularly common in the context of so called the big data, if you're not careful.

There's really a very thin line between data mining and hypothesis shopping. As illustrated by the genome wide associations study example, bigger data sets make it possible to look for increasingly large numbers of patterns. Moreover, the complex machine learning techniques are looking for increasingly complex patterns, which makes it even easier to find something seemingly significant that occurs just by chance.

The techniques we've described are powerful, and their math is sound, but it's important to make sure that you're using them correctly. When used correctly, they let people do things that otherwise wouldn't be possible. But if you don't combine them with the appropriate skepticism and common sense, it's easy to make mistakes and reach the wrong conclusions.

3.2 Deep Learning

3.2.1 Introduction to Deep Learning

In this unit, we're gonna study a revolutionary, exciting, new approach to classification, called, deep learning. In contrast to the approaches we've talked about so far, which are, for the most part, well understood, no one really knows when, or why, deep learning works so well. In fact, truth be told, there's a lot of hype around it, and it's sometimes difficult to separate the facts from the fiction.

What I hope that you'll get out of this unit, is a better sense of what new applications deep learning enables, what makes it so exciting, but also what are the caveats and dangers that come along with it. Deep learning is a major driving force behind many of the recent advances in machine learning, and it's arguably the reason why today truly is the golden age for machine learning.

But there are also many subtle issues lurking underneath the facade. I think it makes sense to start with the excitement, so why is everyone so excited about deep learning? Well it's not shallow learning, so that’s good, but even better, it's let to some remarkable progress on some of the central problems in machine learning.

In fact, it’s even made its way to into the popular press with attention grabbing headlines like, deep neural networks revolutionized the field of speech recognition. Google AI algorithm masters ancient game of Go. Microsoft says its new computer vision system can outperform humans. Deep learning seen as the key to self-driving cars.

There is substance behind each one of these. If you have a smart phone, then chances are, the software on it that translates your voice into words uses deep learning. It's not that speech recognition didn't have good solutions before deep learning came around. It's just that the solutions that came from deep learning are markedly more accurate, and are fast enough that they can even be done in real-time, so that you don't have to wait seconds to have your voice transcribed.

Another major achievement of deep learning, was mastering Go. Go was long thought to be a much harder game for computers to play competitively than either checkers, chess, or backgammon. In each of those games, we've had terrific computer programs for many years. That can beat, or at least complete with, the world's best players.

But until just this year, our best computer programs for Go were nowhere near the top players in the world. And now the crown belongs to the machines. We'll talk more about this example later in the unit. But, what I really wanna talk about are deep learning's applications to computer vision.

Sociologically, that's the example that seem to get everyone's attention. See, machine learning is often driven by challenge problems. There are wide variety of techniques in machine learning, in this module we've seen just a few. And in some other modules in the course, we've covered a few more. But, that's by no means comprehensive.

When it comes to machine learning, everyone has their favorite tools that they bring to bear. But how good are these approaches, compared to each other? Within the computer vision community, there's a famous dataset called ImageNet, at a yearly competition. Let me tell you a bit more about it.

ImageNet is comprised of 1 million labeled images. Each image is labeled with 1 of 1,000 different possible categories. Some of these categories are quite specific, such as a particular breed of dog. Now, this is a really massive collection of images. The goal is to train a classifier that can correctly label images with the smallest possible error rate.

Now there's one technicality, that the way we measure error rates, is to let the classifier choose five different labels, and we count how often the true label of the image in the top five. Every time it is, we call it a success, and every time it isn't, we call it an error.

So what makes image classification so challenging, is that it's something that humans are quite good at, and machines are relatively bad at. The opposite type of problem would be something well defined, like computing products of, say, eight-digit numbers. Surely, your computer or smartphone is much better than you at this, and many other similar tasks.

But there are still some things that humans are very good at, and machines simply haven't caught up to. For those of you who have kids, remember back to the time when your kid was first learning to recognize animals. At first, he may not know the difference between an elephant and a dog.

Maybe he's seen many dogs in person, but never an elephant. So you might point to a picture of an elephant in a story book, and shout, dog! But, it takes a very few examples to teach your child the difference between elephants and dogs. You could show him a few examples of each, and point out the long trunk, and he would be able to extrapolate to all sorts of new pictures that he’s never seen before.

The human mind is amazing, this phenomena is often called one-shot learning, because it takes just a few examples to learn entirely new category. We're quite far away from having machines that can learn nearly so fast. There's been decades of work on computer vision, developing a cornucopia of techniques.

But until recently, the state of the art algorithms can only achieve error rates between 25 and 30% on ImageNet. This is still quite good. If you were to guess five categories randomly for each image, your error rate would be 99.5%. But it's still quite far away from what humans can achieve, which is somewhere in the range between 5 and 6%.

Some of the categories are so specific, that humans get them wrong too. This all changed in 2012 when a team of researchers used deep learning to achieve error rates about 15%. In fact, even these bounds were quickly improved upon. Every year since 2012, the competition has been won by deep learning.

And the entire field of computer vision has shifted its center of mass towards these new techniques. The current best algorithms achieve error rates less than 5%, and amazingly, perform about as well, if not a little bit better, than humans do. This is something that many researchers thought would take decades to accomplish.

But in this problem, and many others, deep learning is rapidly reshaping the boundary of what we think of as possible. It's enabling machine learning in a way that's both awesome and exciting. But it's also a technology, unlike most of what we've covered earlier, that we don't understand the algorithmics behind.

3.2.2 Classification Using a Single Linear Threshold (Perceptron)

Now that we've gotten a sense of the excitement surrounding deep learning, let's get into the mathematics of what it is and what it's all about. Later, I'll tell you about some of the philosophy and motivations from neuroscience behind deep learning. But I think of this as superfluous to understanding on a pragmatic level what deep learning means.

Now is a good time to mention that the word deep in deep learning isn't the judgment call about the intellectual merits of the field. But rather, refers to layering. So, for starters, whenever someone mentions the term deep learning, you should immediately think layered learning. That's what deep learning is about, building complex, hierarchical representations from simple building blocks.

Let's start with the building blocks. In fact, when we covered support vector machines, we already covered what's a common building blocks of deep networks. Remember the classifier itself? That's sometimes called a perceptron. Let me remind you what a perceptron is. A perceptron is a function that has several inputs and one output.

Let's say that it has n inputs just to fix some parameters. Then the output of a perceptron is computed in two steps. In the first step, we compute a linear function of the inputs. The coefficients of the linear function are called weights. In the second step, we take the output of the first step and compute a threshold.

This threshold takes any value above some cut off tao and maps it to the value +1. And maps everything below tao to -1. The second step is the only non-linearity. Now just to make sure that the previous definition was clear, how many parameters define an n input one output perceptron?

While we need n weights and one threshold tao, so in total that's n+1 parameters. Now what's the connection to support vector machines? The class fire that we learned was itself a perceptron. As a class fire it only recognizes linear patterns. And to combat this, we introduced the idea of mapping the input space to a new space using a kernel.

In deep learning, we will take a very different approach where we layer perceptrons on top of each other to get our non-linearities. So let's work up towards that and start with a concrete example. Let's return to the problem of image classification. Now what if we're interested in identifying which images contain a dog and which images don't?

This is already an easier problem than the types of problems that we face in the image net challenge problem, where we even need to distinguish between different breeds of dogs from each other. In any case, as usual, we can represent our input, say a 256 by 256 size image as a vector.

If each pixel is associated with three values, that's, red, green, and blue composition, in what dimensional space should we think of our picture? Well, we can think of it as 256 times 256 times three dimensional space. Now if we want to use a perceptron to solve our image classification problem.

What we're really asking for is a linear function of the pixels that's positive if the image contains a dog, and negative if it doesn't. As you might imagine a perceptron is just too simple a function to solve such a complex task in vision. There probably isn't the nice linear function of the pixels that decides whether or not there is a dog in the picture.

So how can we create more complex functions out of perceptrons as building blocks? Well we can layer them. In the simplest set up, imagine that there are just two layers of perceptrons. Our input is an end-dimensional vector representing a picture. Each perceptron in the first layer has n inputs, each with its own weight.

So each perceptron in the first layer computes its own thresholded linear function. Now comes the interesting part. If we have n perceptrons in the first layer we can have a single perceptron in the second layer, which takes as its inputs all of the outputs of the other n perceptrons.

So how many parameters define this model? Well, we have (n+1)m parameters for the first perceptrons in the first layer and (m+1) parameters for the perceptron in the second layer. What's important is that the overall function we've computed that takes in an n dimensional input factor and outputs a plus one or minus one is much more interesting and complex than the type of function we got from a single perceptron.

We can take this idea much further and have more than two layers. The functions that we get are called deep neural networks, and in practice one usually sets them to have between six and eight layers and millions of perceptrons in between. There are several fragments of intuition behind these types of functions.

The hope is that the lower level layers of the network identify some base features, like edges and patterns, and that each layer on top of them builds on the previous layer to create much more complex, composite features. The intuition is how might you build up a feature that represents whether or not a dog is present in an image?

First you would identify its edges. And then you'd identify which collections of arrangements of edges represent legs and which represent the body and the head and then which arrangements of those parts represent the dog? Recall that in ImageNet we want to correctly classify according to 1,000 different labels at once.

Even though there a million total images, that's not actually that many examples per label. So what's important is that the features that are useful for identifying one breed of dog can be useful in identifying other breeds of dog as well. In this sense, a deep network can have a thousand outputs, one for each label, built on top of a common deep network underneath it, one which is hopefully identifying useful, high level representations that are needed to understand images.

Another rationalization for deep neural networks is that they parallel what happens in the visual cortex. There's still a lot about the brain, okay, most of the brain, that we don't understand. But it does seem that the visual cortex has a similar type of hierarchical structure with neurons on the lower layers recognizing lower level features like edges.

Moreover, we can measure how quickly a human can recognize an object and how quickly a neuron fires. These tell us that even though the visual cortex is performing some hierarchical computation, it requires at most six to eight layers in order to solve high level recognition problems. So let's conclude this discussion by first generalizing what we've talked about so far, and then pointing towards the main topic that we will be interested in for the rest of the unit.

First, the perceptron has a linear and non-linear part, the threshold function. If it wasn't for the threshold, creating deeper networks would not buy us anything. We would still be composing linear functions. And no matter how deep we make the computation, the function we get would still be linear.

It's the non-linearity that we added that makes deep network so functionally expressive. In fact, there are many other non-linear functions that we could have chosen instead of a threshold. We could have chosen a logisti, sigmoid, or hyperbolic tangent, or any other smooth approximation to a step function. This and many other aspects of the architecture of deep neural networks are all valid design choices that have their own merits.

There are many research papers that grapple with issues like which non-linear functions work the best, and how should we structure the internal layers for example, using convolution. We won't talk more about these issues, but it's good to know that they're very important. Second, we've said nothing about actually finding the parameters of a deep network.

Modern deep networks have millions of parameters, which is a very large space to search over. When we talked about the support vector machine, we had the perceptron algorithm, that told us if there is a linear class file, we can find it algorithmically. But even if there is a setting of the parameters of the deep network, that really can classify images accurately, into, say different breeds of dog, how can we find it?

There is no simple answer to this question. There are approaches that seem to work in practice, but why they do it is still very much a mystery and perhaps a phenomenon that has to so with some of the strangeties of searching in such a high dimensional space.

3.2.3 Hierarchical Representations

The nagging question that should be on your mind is, how do we set the parameters of a deep neural network? So far, we've been thinking about them as functions that map a high dimensional vector to multiple outputs. And we constructed these functions hierarchically.

But that only describes the class of functions that we're working with, not how to find the best function for some image classification task that we actually want to solve. Our first goal is to cast the problem of finding good parameters as an optimization problem.

The catch is that not all optimization problems are created equal. There's an important class of optimization problems called convex optimization problems for which we have many good algorithms and we understand quite well, when and why they work. But life is not so easy in the case of deep neural networks.

The optimization problems that come out will be non-convex and unwieldy. We'll talk more about this issue later, but first let's work towards defining the optimization problem we'll be interested in. Returning to the image net example, our deep neural networks have n inputs and 1,000 outputs, one for each category that we'd like to assign images to.

What we're hoping for is that when we feed in a picture as an n dimensional vector, that the output in the last layer contains a 1 In the correct category of 0s in all the other categories. This is sometimes called a one hot encoding. Now what if you give me all the parameters of a deep network? How could I evaluate how good these parameters are?

I could take each one of the 1 million examples in image net, feed each picture into the network and check whether you got the correct label. To be concrete, let me introduce what's called the quadratic cost function. Let's say the input is some vector x and its true label is category j.

Then the output that I would like to get is all 0s except for a a in the jth output bit. Now I can penalize you by how far off your output is from this idealized output. If on input x, you output a1, a2, up to am, as your m output bits, I could take as a penalty aj minus 1 squared, plus the sum of all the other ais squared. This is a function that evaluates the 0. If you get exactly the correct output, and evaluates to something non-zero if you get the wrong category. What we've done is we defined a cost function.

If you give me the parameters of a deep neural network, I can evaluate how good it is by computing the average cost on a typical example from ImageNet. Now if we wanna find a good setting of the parameters, why not look for the setting of the parameters that minimizes this average cost?

This is an optimization problem. We have an explicit function that depends on the parameters as well as the training data that we'd like to minimize. It turns out, that if you find a setting of the parameters that works well on the training examples, i.e., has low average costs, it achieves low error on new sets of examples that you give it to.

This can be made mathematically precise, but it's too much of a digression to get into here. In any case, we have what we're after. We have an optimization problem that, if we could solve, would find good parameters. Is there just some off the shelf way that we can plug in any optimization problem we'd like to solve and get the best answer? Absolutely not.

The difference between what optimization problems are easy to solve and which are hard is one of the foundational issues in theoretical computer science. We don't need to delve too much into that, so let me tell you just what you need to know. First, what types of optimization problems are easy?

Let's start with the one-dimensional case to keep things as simple as possible. What if I give you some function, f of x? And I want you to find the x that minimizes it. Here x is a real variable, so it can take on any real value. There's an important class of functions called convex functions. There are many ways to define convex functions, but let's start with an example.

Take f of x equals x squared. That's convex, now why is it convex? It's convex because whenever you take any two points on the curve, and draw a line between them, the line lies entirely above the curve, except at the end points. That's one definition of convexity. It's not the best definition for our purposes, but it's the easiest one to start with.

A definition that's more natural for us has to do with the second derivative. A function is convex if and only if its second derivative is always nonnegative. Modular issues like what if its second derivative is not defined?

So what's an example of a non-convex function? Let's take f(x) = (x- 1) squared times (x- 2) times (x- 3) as our example. There are points on the curve where the derivative is 0, and it's positive before and negative after. This definitely means that its second derivative is negative around here. The real question is, why is this bad?

The important point is that convex functions have no local minima that are not also global minima. There is nowhere that you can get stuck, if you're greedily following the path of steepest descent, that isn't actually the globally optimal solution. But of the non-convex case, you absolutely can get stuck. In our second example, you could get stuck at x equals 1, which achieves of x equals 0, even though there are xs which make the function negative. Let's abstract what's going on here.

If you have a convex function, and you're at some value x, and you're searching for the value that minimizes f, what you would do is take the derivative at x. If it's negative, you would take a step to the right, increase x. And if it's positive, you would take a step to the left. If you choose the step size correctly, this is guaranteed to converge to the global minimizer of the function.

As you may notice you need to decrease the magnitude of the step based on how large the derivative is. Now what happens if you try the same strategy on a non-convex function? All you can say is that you reach a local minimum, but in general, it will not be the global minimum.

All of these ideas can be extended in a straightforward manner to higher dimensional spaces. Instead of taking the derivative, you would take the gradient. The gradient points in the direction of largest increase for the local linear approximation. And wherever you currently are, you would take a step in the direction opposite to the gradient.

When you have a convex function in high dimensions, this is again guaranteed to converge the globally optimal solution. When you have a non-convex function, it might only converge to a locally optimal solution, or even worse, it could get stuck in a saddle point, which is not a local lothma but for which the gradient is 0.

So what we're going to do is use gradient ascent to find the best parameters for our deep neural network, even though the function we're trying to minimize is non-convex. We're taking an algorithm that's guaranteed to work in the convex case, that we know does not always work in a non-convex case, and using it anyways. One of the greatest mysteries is that it still seems to work. What it finds is not necessarily the globally optimal solution.

But even the locally optimal solutions it finds are seemingly still good enough.

3.2.4 Fitting Parameters Using Backpropagation

Now let's talk about how to apply gradient descent to a deep neural network. We have a cost function that we'd like to minimize. Let me mention a technicality that we haven't addressed so far. When we talked about perceptrons, we described them as a linear plus nonlinear part.

We took the threshold function to be our nonlinearity. But there are many other reasonable choices like a sigmoid function. For what we talk about next, you should think sigmoid. Our perceptrons compute a linear function of their inputs, using their weights, adding a constant term called the bias and feed this value into a sigmoid function to get their output.

The reason it will be important to work with sigmoids is that these functions are smooth and de-frenchable. Or as the derivative of a threshold is everywhere either 0 or undefined. In any case what we've talked about so far is using grading decent to minimize or approximately minimize a non-convex function.

Our functions come from computing the quadratic cost function on the output of the last leg. And it depends on all of the weights and biases inside the network. The gradient is a measure of how changes to the weights and biases change the value of the cost function. And we know we want to move in the direction opposite to the gradient because we want to minimize the cost function.

The main question now is, how do we compute the gradient? In the context of neural networks, we can compute the gradient using what's called back propagation. The equations that define back propagation are quite a mouthful, and involve a lot of matrix vector notation. The intuition is much simpler, though.

What back propagation is really doing is using the chain rule to compute the gradient layer by layer. Let's get into the intution? Imagine in the M outputs of our our neural network are a1, a2 up to am. And these are parameters that we feed into the quadratic cost function.

Then it's straightforward to compute the partial derivative of the cost function with respect to any of these parameters. Now the idea is that these m outputs are themselves based on the weights in the previous life. If we fix the IF output, it's a function of the weights coming into the ith perceptron in the last layer, and also of its bias.

We can compute again how changes in these weights and biases affect the ith output. This is exactly where we need the nonlinear function, in our case, the sigmoid, to be differentiable. Back propagation continues in this manner, computing the partial derivatives of how the cost function changes. As we vary the weights in the layer based the partial derivatives that we've computed for the weights and biases in the layer above it.

That's it, that's back propagation. Now we have all the tools we need to apply deplar. We talked about how to build up complex nonlinear functions from perceptrons as building blocks. We talked about how to cast the problem of fitting its parameters as an optimization problem, albeit a non-convex one.

And we talked about how to compute the gradient in a layer wise manner using back-propagation. Now let me say a few more words about what we've learned so far before we start to apply. Why does gradient descent on a nonconvex function work at all? In lower dimensions, it seems obvious that it really does get stuck.

The truth is that no one knows why it seems to work in high dimensions. There's several possible explanations. Maybe these functions are closer to convex than we think. And are at least convex on a large region of space. Another factor is that what it means for a point to be a local minimum is much more stringent than higher dimensions.

A point is a local minimum if every direction you try to move him, the function starts to increase. Intuitively it seems much harder to be a local minimum in higher dimensions because there are so many directions for you to escape from. Yet, another take away from this discussion is that when you apply back propagation to fit the parameters, you might get very different answers depending on where you start from.

This just doesn't happen with convex functions. Because wherever you start from, you'll always end up in the same place. The globally optimal solution. Also, neural networks first became popular in the 1980s. But computers just weren't powerful enough back then, so it was only possible to work with fairly small neural networks.

The truth is, that if you're stuck with a small neural network and you wanna solve some classification task, there are much better approaches such as support vector machines. But the vast advances in computing power has made it feasible to work with truly huge neural networks. And this is a major driving force behind their resurgence.

Lastly, in our discussion about hierarchical representations. We talked about some of the philosophy in connections to neuroscience. There is at best a very loose parallel between these two subjects. And you're advised to not take this too literally. In the early days, there was so much focus on doing exactly what neuroscience tells us happens in the visual cortex that researchers actually stayed away from gradient descent because it wasn't, and still isn't, clear that the visual cortex can implement these types of algorithms.

As one of my colleagues says, they got caught up in their own metaphor and were missing out on better algorithms just because they weren't neurally plausible.

3.2.5 Convolutional Neural Networks

Let's talk about some of the architectural tricks of the trade. The first and arguably most important is the notion of the convolutional neural network. Let me explain how the first layer works. Let's say that our input is a grayscaled image that's 256 by 256. We can break it up into 8 by 8 image patches.

In total we have a 64 by 64 grid of these patches that comprise the image. Now let's consider a linear function on just the 8 by 8 image patch. Sometimes this is called a filter, in our case a filter is an 8 by 8 grid of weights. And when we apply a filter to an image batch what we do is we take an inner product between them as vectors.

So if we apply a filter to each one of the patches we get a new 64 by 64 grid of numbers. What good is this? Remember the intuition is that the first few layers detect simple objects like edges. We can think of a filter as a naive object detector.

And instead of having it have different parameters when we apply it to each of the different image patches. Why not have the same parameters throughout? This drastically reduces the number of parameters and consequently there are many fewer things to learn. If we take this idea to its natural conclusion we get convolutional neural networks.

In the first layer, we have a collection of filters, each one is applied to each of the image patches. And together, they give the output of the first layer after applying a non-linearity at the end. This is already a major innovation because it means we can work with much larger neural networks.

In practice, just the first few layers are convolutional, and the others are general and fully connected. Another important idea is the notion of dropout. Here when we compute how well a neural network classifies some image, say through the quadratic cost function. We instead randomly delete some fraction of the network and then compute the new function from the input to the outputs.

The idea is, if a neural network continues to work even if we drop perceptrons from the intermediate layers. Then it must be spreading information out in a way that no node is a single point of failure. Training a neural network with dropout makes the function we learn more robust.

This is not a precise statement, but it's the prevailing intuition. Now, you know the secret sauce of deep learning. Sometimes it's hard to wade through what's fact, what's fiction, and what's hype. We've done all the hard work and laid out the foundations. And now it's time to have some fun.

In the remainder of the unit, we'll cover some of Deep Learning's truly amazing applications.

3.2.6 Case Study – AlphaGo

As we discussed earlier, machine learning is driven by challenge tasks. For example, can we automatically classify images as accurately as humans can? Deep learning and convolutional neural networks in particular have been a major driving force behind why the answer to this question has changed from a resounding no into a yes.

What about games of strategy? Can we learn to play as well as the best human experts? Board games in particular have long been a fascination for the field of machine learning. In 1992, researchers developed a program to play checkers that dethroned the world champion. In 1997, IBMs Deep Blue played six matches against chess grand master Garry Kasparov and defeated him by a score of three and a half to two and a half.

Kasparov said that he sometimes saw creativity and intelligence in the moves that it made. The last game to fall was the ancient game of go, invented in China over 2500 years ago. We'll talk about what went into this breakthrough in detail later. First, let's develop an important concept called the game tree.

Let's think about it in the context of a simple game like tic-tac-toe that you're all probably familiar with. The game tree is just the way of describing all possible configurations of the game and how they relate to each other. In the first level is the starting configuration. No one has placed any Xs or Os.

The first player has nine positions where he could play and each one of these choices leads to a new configuration. We think of these as the second level. So the configuration at the first level connects the nine configurations on the next level and taking it further for each configuration on the second level, how many configurations on the third level can it reach?

Well the answer is eight because one position is already occupied and there's eight remaining places that the second layer could play. Now the key is that at the end of the game either we have a winner or the game is tied. Let's build up some intuition for what the optimal strategy looks like.

Imagine we're at some configuration and it's the first player's turn to move. Furthermore, suppose that all of the configurations that can be reached based on his move result in the game being over. What should he do? Well, if there's any move that result in him winning, he should choose it.

But of course, they might not be any such move. So if he can't find a move that wins him the game then, he should instead choose a move that results in a tie. And there's no such move either then, there's really nothing he can do, he's gonna lose regardless of what he chooses.

But now we know exactly how good this configuration is. If there was a move he could make to win we should think of the configuration we're currently at as a winning configuration too. It's as if when we reach it, the game is already over because we know exactly what the first player would do next.

Similarly, if the best he could do is tie, we should think of the configuration as being a tie, by the same logic. And otherwise, it corresponds to a loss. What we're doing, is marking, an intermediate state of the game based on how well a player would do if he played optimally from then on out.

In this way we can back propagate the information from the end of the game to the beginning. Of course, nothing I've said is particular to tic-tac-toe. This is how many games work. There's some space of configurations that are connected based on how choices the players make move from one configuration to another.

What I described is called backwards induction. From the end of the game where you know the outcome, you can figure out how good each intermediate configuration is too, if you and your opponent were both to play optimally from then on out. Tic-tac-toe is a straightforward game to play.

But checkers and chess are much more involved. Let's talk about using machine learning to play chess. The basic idea is the same. A configuration corresponds to an arrangement of pieces on the board. Now the catch is that the space of all possible configurations is gigantic. We could never hope to explore this space using backwards induction.

We have no chance of finding or even concisely describing whatever the optimal strategy is for chess, even though we know it exists. So instead what we need is some way of stopping short in the game tree, and evaluating how good a configuration is without exploring every possible configuration beneath it.

What we need is some proxy or prediction of who's going to win. Now, for chess, it's not that hard to come up with a rough measure of how good a configuration is. When you're first taught to play chess, you're taught that a bishop is worth 3 pawns, a rook is worth 5 pawns, and a queen is worth 9 pawns, and so on.

This gives us, already, a naive measure. In some configurations, we could ignore where the pieces are on the board and just add up the value of the remaining pieces I have and compare it to yours. We could use the ratio or some other measure to estimate our respective odds of winning.

You can construct a naive chess playing program in the following manner. Brute force search over some number of levels, these correspond to numbers of moves you're looking forward into the future. And then you evaluate the estimated probability of winning at each configuration. And use backwards induction to find your best next move.

Such a program wouldn't beat Garry Kasparov. He's much more impressive than that. But what went into the IBM Deep Blue system follow the same design principles. The difference is in how deeply evaluated a configuration. It took a more nuanced approach. If a pawn is blocking a rook then it restricts one of its directions of motion.

Such a rook should be worth slightly less than it would be otherwise. But if that same pawn is able to take another pawn diagonally, then it can move out of the way, and it's called the transparent pawn. The rook is now worth almost as much as if there were no pawn in front.

There are thousands of such rules that play into how to properly evaluate a configuration. These were hand-built into Deep Blue, and discovered by consulting with chess experts. This is how Deep Blue won. But Go is very different. I won't describe the rules of go, but instead I'll say that it's a game about occupying as much territory as possible.

How the different battlefields interact, and what properties of their shape determine who is doing better are very complex. There isn't even a simple rule that gets things off the ground. Until recently computer programs for Go could be amateurs but never experts. It was believed that we were as much as ten years away from Go programs that could even compete with experts.

But in March 2016 Google's program Alpha Go defeated Lee Sedol, the current European champion, four games to one. There were a number of eyebrow raising moments and ideas that went into it. At the heart of the approach, was the idea of using a deep neural network to evaluate a board configuration.

This is something that in chess required thousands of hand designed rules and in Go seemed unmanageable. At one point in the matches, a panel of expert commentators were convinced that the match was roughly even. Yet in Google's control room, AlphaGo was confident that it was up by about 12 stones.

In chess the goal was never to evaluate a board configuration as well as a human. The goal was to do almost as well, and then use computing power to explore more of the game tree. In Go, it seems like deep neural networks evaluate board configurations differently than expert humans do.

And given that AlphaGo won I'd say it's probably correct, but how did we get to this point? AlphaGo was trained on hundreds of thousands of games by expert players. And it was able to predict the next move with accuracy 57%. But its ultimate goal was not just to play the same moves as humans, but actually to play better.

To accomplish this it was pitted against itself many times. All the while refining the parameters of its deep neural network which had used to evaluate board configurations. What's truly amazing and also a bit frightening is that it learned some beautifully creative moves all on its own. At one point in the match against Lee Sedol it made such an unconventional play that some of the commentators thought it must have been a bug in the code.

It was a strategy that simply put, no human had ever thought of in 2,500 years. Lee Sedol was so taken aback by it that he had to leave the room for 15 minutes to collect his thoughts, slowly appreciating the beauty and the strategy behind what, and why, AlphaGo had done.

This move was widely regarded as the pivotal point in the game, which AlphaGo went on to win.

3.2.7 Limitations of Deep Neural Network

I hope that I've given you a sense of the excitement surrounding the networks. But now it's time to take a sobering look at a more frightening aspect. In the examples we've seen, deep learning has been able to conquer some very hard challenge problems. And that's a great reason for being optimistic that they'll lead to practical self-driving cars very soon and all sorts of other technological advances.

But there's an important worry. When we talked about fitting a deep network's parameters, we discussed the difference between convex and non-convex functions. The truth is that we don't really understand why things like gradient descent work so well and even then, the actual features that it finds are difficult to interpret.

We can't readily take a deep neural network that we've learned and see why it's working so well. This leaves a lot of room for manipulation. Following the work of what if we start with an image of a deep network successfully classifies? How much do we have to change the image to change the label that the network assigns?

Let's say I give you an image and a target label, say ostrich. I want you to fool the deep network into thinking that the image is an ostrich, even though it's very clearly not. As usual, we represent the image as a high dimensional vector x. Now we can look for the smallest vector R, which we think of as a perturbation to the image.

So that when the deep network is applied to X plus R, the output has a 1 in the ostrich category and 0s in all the others. Finding such an R is a non-convex optimization problem. But, as usual, there are approaches for approximately solving it, nonetheless. Obviously, if I'm allowed to change the image entirely, then I can make the deep network think it's an ostrich by making the picture be of an ostrich.

The question is, how small can the perturbation be? It turns out that you can get away with the perturbation that's imperceptible to the human eye. Now let's put that into perspective. Much of the initial excitement of deep learning came from the fact that now we are able to classify images automatically with error rates that are even slightly better than what humans can achieve.

But the classification function that they've learned is so discontinuous and unwieldy, that changes that ought not to change the category. In fact, you wouldn't even see them as a change to the image at all, can be used to fool the deep network into thinking the image is whatever category you like.

In fact what's even more frightening is that the same perturbation works across many neural networks. So you could think that if I don't know what neural network you've learned, and don't have access to the millions of weights and biases that define your model, I won't be able to fool it.

But it turns out that I could learn my own neural network even using a subset of the training data that you've used, find the perturbation for a given image that makes my network think that it's an ostrich, and the same perturbation would work for yours too. So all this time we've been talking about deep neural networks being able to compete with, or even out perform humans.

If they can find new moves in the ancient game of Go, then why not solve other sorts of decision making problems for us? We could imagine relying on them to keep us safe on our daily commute in self driving cars. Or diagnosing us based on our symptoms and medical history.

Or even estimating the credit risk of an individual by incorporating more information than currently goes into your FICO score. But all of these wonderful possibilities come with a caveat that if we don't understand why deep learning works, the classifiers we learn might be subject to all sorts of nefarious manipulations.

Or might learn unfair and uncontrollable biases in how they make decisions.

Module – 4: Recommendation Systems (Philippe Rigollet and Devavrat Shah)

4.1 Recommendations and Ranking

4.1.1 What Does a Recommendation System Do?

Welcome to this module on recommendation systems. I am Philippe Rigollet from the Mathematics Department and the Center for Statistics at IDSS. A large part of my research is devoted to high-dimensional statistics, and I also teach courses on this topic at MIT. In a nutshell, the goal is to find a needle of information in a haystack of data.

As we will see, the goal of recommendation systems are similar in many ways.

Hello, I'm Devavrat Shah. I'm with the Department of Electrical Engineering and Computer Science at MIT. I'm member of the Center for Statistics at ITSS Institute for Data Systems and Society. Philip and I will be your co-instructors for Modular Recommendation System.

My background is in machine learning, and large scale data processing. I teach courses on data topics in MIT regularly. I have spent the past decade thinking about extracting meaningful information from social data, the data that we generate. As we shall see in this module recommendations system represent an important large class of societal questions that precisely deal with extracting information about people's' preferences.

What people like, what people don't like, from the data that they provide. The goal of this module is to discuss radius challenges associated with designing recommendation system from both statistical viewpoint, and computational viewpoint.

Before we dive into the subject, we should answer an important question if ever.

What is a recommendation system?

Great question, Philip. So in a nut shell, a recommendation system is something that helps you find what you're looking for. I see, like a search engine, for example, Google. Great point. It's a little different than that. You see, in Google, when you research something, you need to know exactly what you want to look for.

For example, if there's a restaurant that I know precisely that I wanna go for dinner tonight. All I will do is in Google search bar I will type the name of the restaurant and then boom. Google will give me the address of the restaurant, the menu of the restaurant, the hours of the restaurant, and what not about the restaurant.

I see but what if I don't know what kind of restaurant I'm looking for? Google is not making any recommendations to me.

Great point again, Phillip. See, Google search is helping me find answer to the question. When I know precisely what question I'm trying to and get answer.

Recommendation system is less concrete. It's a bit mysterious you see. Let's take an example of Yelp for that matter. Yelp is perfect for recommendation because it's suppose recommend restaurants to us. So let's say hypothetically we are new to this great town of Cambridge where MIT is, and we need to go for dinner tonight.

Great, let's find a good barbecue.

Well, if you forgot, I'm a vegetarian.

How about we try American, or whatever that is?

Okay, American cuisine it is.

All right, so let's find an American restaurant nearby using Yelp and that fits our budget.

In a sense, a recommendation system is also a search engine. When I don't know precisely what to search for.

Absolutely, Philip, and here search is more semantic. See, the semantic associated with the search intent is up to the interpretation. It's a context dependent. To a large extent, it's very personalized. It's personalized depending on the person who is searching for the answer.

For example, what a good restaurant might mean to you, it may mean not so good to somebody else. Or what a good restaurant, to you today, might mean something different yesterday, and might mean something different tomorrow.

In a sense a recommendation system is also a search engine when I don't know precisely what to search for.

That's absolutely correct. The search here is more semantic. The semantics associated with this search intent is up to interpretation. It's a context dependent and to a large extent it's very personalized to the person who is searching. For example, what a good restaurant might mean to you, may not mean good to somebody else, like me.

Or what a good restaurant might mean to you today, might mean something very different yesterday, and might mean something very different tomorrow. It is this semantic aspect of the search that makes designing a recommendation system much, much harder than building just a search engine.

So see, here is the top American cuisine restaurant in the area according to Yelp users. So see, for example, I don't like this restaurant. I think it's too loud and it's too expensive. Sorry, but I guess I'm not your typical Yelp user.

Indeed, that's exactly what I meant about personalization a few seconds back. We'll see that it's a key feature of powerful recommendation systems. The recommendation is different for each and every user. Customized to ones taste and is different depending on the context be it time, place, or even ones mood.

That's correct. You see this when you shop on Amazon. The site makes recommendations that are based on your past purchases. In my case, I'm a huge grilling fan. It recommends barbecue items all summer long.

The same applies when you watch TV or other media. The Netflix recommendation system has ordered the catalog for you so that you don't have to search forever. A recommendation system is the primary reason for the success of Netflix as a business. They have managed to make it's users feel as if there's an infinite pool of catalog, even though, actually they're about a really small catalog.

And music too. It's pretty hard to keep up with music trends these days and recommendation systems help you do just that. It tells you what music you might enjoy listening to. For example, systems like Pandora or Spotify provide excellent recommendations. I've already discovered this band I like called The Arcs using Pandora's recommendation system.

But sometimes recommendation is not just for convenience. But it is necessity. Take for example of YouTube. You know how much content that's from YouTube generates? Every minute, here's the statistics I came up with. Every minute, 300 hours of video is uploaded on YouTube. So let's hypothetically assume that you and I are going to 100% of our time continue watching YouTube. That it will take a million lifetime for us to watch content generated in one lifetime.

In a sense, recommendation system is essential to discover the content that we care about on YouTube otherwise we would not be able to deal with it

4.1.2 What Does a Recommendation System Do?

So is that all? Do recommendation systems help me find things that I might want to consume?

No, no, we're just scratching the surface, it's tip of an iceberg. You see, recommendation system is a very large class of prediction problem. At the core of it, it's trying to understand an individual's choice or preference or personalized preferences for that matter.

So wherever it is crucial to understand once choice, things you like, things you dislike. Recommendation systems are very useful. For example, consider online advertisement. Like today morning, I visited my New York Times website. So, when I was in New York Times website, I was reading my content, but hey you know what, on the side of it or sometimes in the middle of it, they showed me an advertisement.

So, now the question is that, what advertisement should they show it to me that is already personalized. And, this is a recommendation problem.

Absolutely. And in this scenario, the question is about determining what you as a customer is going to be interested in the most in order to target advertising.

This is actually very similar to the problem of determining what you might like to eat, read or watch, a problem of recommendations. One way to look at a recommendation system is that it's trying to match people to products, restaurants, movies or ads. But more generally, it could match people to people just like in a dating system.

Think of the website eHarmony for example. Here people are seeking partners to date and eventually marry or never see again in some instances. The goal of eHarmony is to match two people who are likely to be attracted to each other. We do not know exactly what eHarmony does but in principle a system like this is trying to do the following.

Consider an eHarmony customer. Let's call it Devavrat. Sorry, that was unfortunate. So Devavrat provides information to the site, mostly about himself, but also about his preferences. Using this information, the system tries to figure out his preferences and find potential matches in the database. Now, Devavrat may have very high standards.

And among his matches, the system also has to find those for whom Devavrat is, in turn, a match. And if we can achieve this perfectly, it's truly remarkable. Most of the time, people are like Devavrat, they don't even know what they are looking for.

Totally, I did not know what I was looking for til I saw my wife, Salvia.

Look, I got married. Now Tinder, it's another system like this, another dating system of the modern time. The way it works, at least in principle, is as follows. Tinder is an app. You enroll into it and then you are shown pictures of other Tinder inhabitants who are in your geographic proximity.

You swipe to left or right depending upon whether you like the person or you don't like the person. And the person on the other end, if you like the person, gets notification. In principle, Tinder would like to show you exactly those people that you like. So you're always swiping on one side.

But of course, that doesn't happen. But it's trying to learn this by asking you question. And the brilliance of this system is, the way they're asking you the question, they're incentivizing you to answer them correctly. You see, you want to answer the likes by sort of the ones you like and the ones you don't like.

Now Philip as a Tinder expert, let me give you advice. Get a better headshot so that your wife will like you more.

How about something like this?

Works.

Or maybe the intellectual. All right, so we know that recommendation systems are useful for finding people's preferences and use them to match them to the choices that they may prefer. And that's it right?

Actually no. Recommendation systems are like a bottomless pit. They have far reaching implications in the operation of work. Take any social systems that you can think of. Be it infrastructure like transportation, e-commerce, brick and mortar retail, entertainment, media, and even financial institution. They are all affected by how people make choices.

What people like, what people dislike. After all, it's about people's choice that determines demand. And as we all know from basic macroeconomy, there are two sides of economy. One is supply, and another is demand. So if we can understand demand really well, then we can actually plan the supply well, and that will make, sort of, operations better.

So as a matter of fact, consider one of the largest operational problem of our time. The logistics, or managing supply chain. A supply chain that takes goods from the hand of manufacturer, from say somewhere far away in China, and brings it to my, or your doorsteps. This is the key operational problem that businesses like Amazon, in United States, Alibaba in China, Flipkart in India are grappling wit.

On one hand, as we have discussed in depth, the ability to solve the right product recommendations on e-commerce website helps consumers make the right purchases. On the other end, those exact interactions can help them inform what are the things that people like and dislike and hence manage supply chain slash manufacturing really efficiently.

Wow, so it looks like recommendation systems really are everywhere. Any other aspects that we should discuss?

Actually, it might be fun to discuss that Tinder example again. Remember, we are Tinder veterans, without real Tinder accounts. Recall the movie Social Network a la Facebook, Zuckerberg. Remember the game of Hot or Not?

So, in an interactive world, where individuals are available to engage fully, one way to think of a recommendation system is like playing a game of 20 questions. I want to know what you are thinking, and I want to know that by asking as few questions as possible.

Absolutely, it's like playing the game Guess Who against the Gary Kasperov of Guess Who?

He guesses right in a record number of questions.

Bingo, this is exactly how a recommendation can work in an interactive situation. Later in this module we will discuss a case study related to selecting beers and gathering opinion of people using comparison like questions and answers.

4.1.3 What Does a Recommendation System Do?

Tinder, beer, and Guess Who are all fun and games. But recommendation systems are also very serious business. They have close conceptual ties to the core of a democratic society, such as how to run elections or elect governments based on collective opinions. This is called Social Choice Theory.

How does it work? We have discussed enough example of what the recommendation system is and its utility. Now the question is how do we do it? How do we go about building it? Are Tinder, Pandora, and the Amazon recommendation system built on the same principles?

Well. recommendation systems are age-old.

They way they work historically is by we calling up our friends, people we know, and whose opinions and preferences we trust and ask them for suggestions. At the core, modern recommendation systems are not much different from this. They are automated, they operate at scale, and hence require some sophistication.

Very crudely speaking, to find what you might like we can find other people with similar taste, similar interest, similar opinions, and then use that preference or dislike, what I might like or I might dislike. Such algorithms are known as collaborative filtering. We filter or predict our preferences through collaboration.

We shall spend quite a bit of time in this module discussing various algorithms for recommendations based on this very basic insight. This insight, when viewed mathematically, leads to unusual algebraic approaches for recommendation. It might be worth cautioning while the insight described seems very simple, it takes a lot more to actually building a real system, and we shall discuss all of this as part of this module.

So before we go into details, lets discuss a few of the nuances involved in building a recommendation system.

Transparency of recommendation systems are important. When systems like Amazon product recommendation chooses to display an item, it is important for the user to understand what that means. For example, it is important to tell them that this is the item that they may want to buy together with another item.

If you're buying an electric snow blower, make sure that you get a long extension cord as you will need that. When a recommendation for such an extension cord is made, it is important that the user understand why this item is recommended. In this case, that the cord was frequently bought together with the snow blower.

Similarly, When you are recommended a movie, like Goodfellas, you may want to know that you are suggested it because you watched and liked The God Father, not because you watched Sponge Bob with your three year old.

Simple machine learning doesn’t work well for recommendation. Remember Pandora, the music streaming company we spoke about earlier?

It works like this. You create a virtual station on Pandora by suggesting the name of a band, or a song, or whatever. And then Pandora keeps streaming music that corresponds to that station's flavor. Initially when Pandora started, they were trying to build such a system without really using recommendation system data.

They didn't have any to begin with. Instead they had specialists, specialists that annotated songs based on musical characteristics such as electric guitar riffs or minor key tonality and whatnot. And then in a blink a very good system that thought that BeeGees was like Beatles. Not sure if BeeGees fan, or for that matter Beatles fan, would agree with that.

Looking under the hood, they realized that the reason behind such a mess was very simple. In early part of their career, the BeeGees were aspiring to become like Beatles, in fact they were a Beatles knockoff.

Now, of course, if we looked at the people who listened to those popular BeeGees songs, it would be very different from those who listened to The Beatles.

And this suggests that the importance of behavioral data in building recommendation systems, it's truly social. To know your preferences, you don't really need to understand why you listen to The Beatles, but find out who also likes The Beatles, and see what else they like.

But behavior data is not easy to get. Think of Netflix on day one.

Or Pandora on day one, for that matter.

When you do not have any behavioral data, it makes it really hard to get going, or you do really poorly in the beginning. This is known as the cold start problem. There are many ways to get around this, and we shall discuss this aspect later in the module.

Now, as we know Pandora has gone on to become a very successful recommendation system. So, they really did solve the cold start problem.

Self Fulfilling Prophecy. Major flip side of recommendations is that it may not allow for discovering new things by chance. Effectively, a good recommendation system will keep recommending things that are liked or preferred by most, and create a herding phenomenon.

Newcomers, who are really good, might not get any attention at all, this is a typical self fulfilling prophecy phenomenon. We'll discuss how to avoid this in a principal matter in this module.

Recommendations are means to an end. Looking for a suitable person to date or marry, when you don't know exactly what you are looking for, or a new restaurant to eat, movies to watch, songs to listen to, or-

There are businesses opportunities. Targeted advertising, product purchase suggestions and what not.

Recommendations should be objective driven. Advertising to increase click through rate. Product recommendations to increase conversions. Recommendations for profit increase might be very different from the recommendations that you provide for customer satisfaction.

Building a real system is a lot more complicated. Building a real recommendation system requires a lot of context specific system design in addition to all the basic principles that we will learn in this module. So if you have to build a recommendation system that works for you, this module will provide you with all the essential knowledge that you need In addition to all the knowledge that you have from your domain context or application.

When put these two together, you will have a functional, high performing, recommendation system.

4.2.1 So What is the Recommendation prediction Problem? And what data do we have?

Now that we have a good idea of what recommendation systems are and where they are used, let us look a little closer at the data science questions behind them. Specifically, what does the data look like and what do we wanna do with it? It seems that a simple approach would be to consider this problem as a binary question.

Given a pair, a user and a product, should I recommend this product to this user? Yes or no. Classification is perhaps the best known learning problem and specifically answers yes, no questions. We could just use our favorite classification method such as support vector machines or boosting, for example, in the context of recommendations.

Actually, we're gonna be able to do much more than just giving a yes, no answer. We're going to be able to rank products and even guess how the user would rate them. Let us do the first thing that we should always do in data science, take a closer look at our data.

Netflix, the video streaming service, is a name that is strongly associated to recommendation systems. Clearly their goal is to recommend movies to their users, but the main reason for this is that in 2006, Netflix did a superb PR coup. They organized a competition with a grand prize of $1 million to the first person who would improve the performance of their current system by at least 10%.

While this may seem like a lot of money, it was actually a rather economical way of getting some of the sharpest minds on the planet to work hard on their problems. Academics, researchers, engineers, students, or even hobbyists, thousands of teams enrolled. It was a huge success, and it started the whole data science competition business.

The problem was simple. Netflix released a data set of how users had rated movies on a scale from one to five, about 100 million triplets of the form user, movie rating, were released to the competitors. It was effectively a giant list of the form Bob rated Goodfellas, four, Emily rated Titanic, three, Frank rated Dirty Dancing, five.

This Frank, he really likes Dirty Dancing. This is called the training set and competitors used it to calibrate or train their algorithm. Along with this set, Netflix also released another million incomplete triplets of the form. Ted rated Rio Grande, Soledad rated Deliverance. The goal is to predict hidden scores.

These were real number scores, integer numbers 1, 2, 3, 4, or 5. But the predictions could be any number, like 4.52. This was useful if your algorithm couldn't decide between a four or a five, it could mitigate the losses. The performance was measured in terms of what is called root, mean, squared, error.

Let's parse this term to define it. Let's say that a competitor predicts 3.4, 4.1, and 2.0, but the true scores are respectively 4, 2, and 3. The errors are 3.4 minus 4 which is -0.6, 4.1 minus 2 which is 2.1, and 2.0 minus 3 which is -1.0. These can be negative and cancel each other, so we square them to get the squared errors.

0.6 squared is 0.36, 2.1 squared is 4.41, and 1.0 squared is 1.0. Then we average them to get the mean squared error. 0.36 plus 4.41 + 1 divided by 3 equals approximately 1.9. Finally, the root mean squared error is just the square root of the mean squared error.

Square root of 1.9 is approximately 1.38. The last square root operation is not very important, but it allows for the following interpretation. In average, the algorithm made an error of about 1.4 per user movie pair, which is not very good in this case actually. In 2006, the state of the art Netflix system had a score of 0.95.

And the winning team dropped this number down to 0.85 after three years of hard work. Once the missing numbers are predicted we, can recommend movies using simple rules. For example, we could recommend a movie to a user if the predicted score is at least 4.2. Towards the end of this video, we'll see what else can be done with this predicting numbers, and if there is something else we should be trying to predict.

4.2.2 So What is the Recommendation prediction Problem? And What Data do we Have?

Let's get back to our data. 100 million triplets is huge. And remember that this is only what Netflix released for this competition, but the data set of all their users and all their movies is actually much, much, much larger. Let's try to organize this data a bit.

The most natural way to do this is to put these numbers into a table, where the rows are users, and the columns are movies. The cells of this table contain the ratings. The Netflix price data set had about half a million users and 17,000 movies. So we should build a table with half a million rows and 17,000 columns.

Here is the row corresponding to Frank and here is the column corresponding to the movie Dirty Dancing. This entry of the table is Frank's rating of Dirty Dancing, a five. Let's see where our 100 million scores look like in this table. It's a drop in the ocean. We know about 1.2% of the entries of the entire table, and the remaining entries are completely unknown.

To be fair, I should say that in the case of the Netflix prize, the goal was to predict only a few missing numbers, about 1 million still. But Netflix needs to predict all numbers every day. This looks like an impossible task, right? How does knowing that Frank is a Dirty Dancing fan, can help me predict how Ted is going to like the western, Rio Grande?

We will see how our statistical modeling coupled with efficient algorithms, can help us extrapolate known ratings to the entire table, by leveraging realistic structural assumptions. In mathematics, a table filled with numbers is called a matrix. This is not to just sound fancy. Matrices are a well studied object in the field called linear algebra.

Using this theory, we can not only describe mathematically our structural assumptions, but also leverage a powerful algorithmic toolbox that can be scaled to the size of the whole Netflix data set, and even larger. It turns out that the tinder data set is essentially the same. Men are rows, here's Frank.

Womens are column, and the entries are binary here. 0 if swipe left, 1 if swipe right, and then a lot of question marks. The eHarmony data set is essentially the same, but the meaning of the entries is not as clear. It's some sort of a compatibility index. It's computing using side information.

This is the first thing that a dating site asks you, when you subscribe. Answer a long list of questions to collect informations about you. They can even extract some of your physical features, or type of activities from the photos you post. Exploiting this site of information requires a lot of domain specific knowledge, and this is a whole field in itself.

In a way this is the secret sauce of eHarmony. Let's go back to our Netflix example. Netflix knows a bit about their users, location, type of credit card they used at the time of subscription, their browsing history, what type of smartphone they use and a lot about movies.

Cast, the director, year, rating, budget, etc. Leveraging this information has been instrumental to win the Netflix prize as well. To conclude this video, let me come back to a remark that I made earlier. Netflix may be interested in other objectives than simply recommending you a movie that you will like.

For example, Netflix may be interested in showing you as many summary pages as possible to collect even more information about your browsing pattern. To do this, all the numbers fill the table, not just the ratings. Here's another example, Amazon may be interested in recommending a product on which they expect to make more profit rather than the one you are the most likely to purchase.

For example, assume that the chances that you purchase coffee machine A or 20% if recommended to you and 10% for coffee machine B. However, Amazon receives $100 each time they sell coffee machine B versus only $10 for coffee machine A. In expectation, Amazon will receive $10 times 20% equals $2 for coffee machine A, versus $100 times 10% equals $10 for coffee machine B.

So Amazon may be tempted to recommend coffee machine B, rather than A. Unlike Amazon or Netflix, who are matching users to products, Tinder or eHarmony are matching users to users. The key difference is that products don't have any preferences, but users do. In other words, Tinder and eHarmony have a more complicated problems on their hands, and we'll touch-up on the specifics later in this module.

To account for these different objectives, a good data scientist must know what values to enter in the table. Chances of buying or expected revenue, for example, this is a gain domain specific, and often several recommendation system are run in parallel to recount for these objectives. To summarize, we've seen that the data can be thought of as a matrix with lots of missing entries.

We need to fill out these entries. The meaning of these entries is very much context specific. In the next few videos we'll see how we can make simplifying assumptions that can help us solve the problem.

4.3.1 Recommendation Algorithm 101: Using Population Averages

Welcome back. Today's topic is going to be the first recommendation algorithm. Thus far, in this module, we will spend time discussing 1, what a recommendation system is. 2, where a recommendation system can be useful. And 3, what is the basic problem of recommendation. As we have seen, in its simplest form, the problem of recommendation system is filling missing entries.

For example, we can think of the available data as a sheet of an Excel file. So you've got the rows, and you've got columns in your Excel file. In the context of Amazon product recommendation, rows would correspond to customers, and columns would correspond to products. In the context of Netflix, rows would correspond to, again, customers, columns would correspond to movies, and the cells or the entries in the cells would correspond to the ratings.

In the context of Yelp, it is about whether a customer is going to be interested in eating at a restaurant or not. In all set scenarios, a large fraction of entries are actually missing. After all, each customer ends up eating at a very small fraction of restaurants in a town.

So if we threw a dart at random on this big Excel sheet, It will most likely end up on an empty cell. So really, that says data is really sparse. In the next few chunks, we shall discuss algorithms for filling these missing entries. Now to fill these entries, we will leverage structure in the data.

In today's module, we'll start with the simplest possible structure, and then evolve to more complicated structures as we go along. The simplest possible structure, the entire population thinks alike, eats alike, sleeps alike. A bad assumption in general, but excellent place to get started on this long journey of designing recommendation algorithms.

The resulting algorithm is effectively what we shall call population averages. Let's start with our classical Yelp scenario. Remember, we were looking for a restaurant to have dinner, and Philippe and I settled on having an American cuisine. So let's assume that you are going to go to Cambridge or anywhere in Boston, for that matter, to have American cuisine.

Let me add a twist to it, you want to celebrate an occasion. You have successfully gone through first few chunks of recommendation system, and so you're happy to go to an expensive restaurant. So now, you search, various restaurants come up as a result. Just like a search result, but filtered by your preferences.

There are too many restaurants qualifying your constraints. So now, what? Well, you need to order them somehow. And this is where recommendation system becomes really useful. And this is precisely what Yelp does. It provides options for you to rank your results or alter them. One of the options is by the proximity, which is not really relevant, because remember, we are happy to go anywhere.

Another option corresponds to how pricey the restaurant is. Again, we're keeping only the most expensive restaurant, not sure it is relevant either. At this stage, we are looking for what restaurant is really good. Yelp is ordering for this by suggesting order by rating. Great, various people who have been to these restaurants have provided ratings to these restaurants between one to five stars, and one can use these ratings to produce ranking for restaurants.

And this is what Yelp is doing in this option. So one way to compute these rankings is as follows. 1, for each restaurant, assign it a score, a score which is equal to the average value of the ratings that it has received, okay? So for example, if a restaurant has received two ratings, five and one, then its average would be five plus one equals to six, divided by two, which is three.

2, Order restaurants as per their score. So restaurants with higher score gets higher rank, restaurants with lower score get lower rank. That is it. In a sense, this is what Yelp is doing when it's showing you that option of order by ratings. Super easy. If you are building a system, this can be obtained using a native SQL like database queries.

In other words, very easy to implement and scale. In our Excel sheet visualization, effectively, we are filling out missing entries for a given cell by taking the column average of the entries. In practice, one has to be a little bit more careful than just taking averages. To understand this, we will have a small lesson.

So suppose you have a coin, a coin like this, which has two sides. It has a head, and it has a tail. Every time you flip it and toss it like this, it comes up, it lands in my hand, and either it's head or it's tail. Now you want to understand, what is the chance that this coin, when I tossed, comes up head or tail?

Now, a general belief for all of us is that, well, a coin is built symmetrically, so it will be 50% chance it will be head, 50% chance that it will be tail. But that may not be necessarily true. Somebody might have rigged it, maybe the physics of the coin is different, who knows?

So maybe you and I want to figure out exactly what the bias is. Now, in this case, we can verify. How can we verify? Well, we can measure it. We can toss this coin many times and observe its outcome. So when I toss a coin once, it comes up head, coin toss twice, tail, head, tail, etc.

Now, while I'm running this experiment, at any point, we can state the chance of a head showing up, right? Because I can say, in so many tosses, let's say 100 times, I have observed 46 times heads, which means that 46% chance for it to be sure it's coming up head.

Every time when I toss a coin, and it comes up head, we are assigning score 1. Every time tail shows up, we assign a score of 0. In that case, all I'm doing is I'm computing average score. Now mapping it back to our Yelp setting, if we think of head as like or 1, and tail as dislike or 0, then effectively, average score of a coin, which is the same as computing average rating of the restaurant the way we discussed sometime back.

4.3.2 Recommendation Algorithm 101: Using Population Averages

Now, let's consider this setting. In this setting our intuition suggests that this way of measuring the chances of head is the right thing. And indeed in probability theory this is known as the Law of Large Numbers. If we toss a coin enough times then the measured score of the coin becomes arbitrarily close to the actual chance of seeing the head.

So this way of measuring the bias or the score of a coin is a perfect thing to do. Maybe there was a reason why we considered the population averages as an algorithm to begin with. But this is good when you have a lot of data. It is assuming that there's a lot of ratings in the case of restaurants that are available.

In reality we have very limited data for some restaurants. Consider a newly opened restaurant. It might have only a few ratings. Therefore, we may need to worry about what happens to such an estimator when we have very few observations. Back to coin setting, suppose we have only one observation.

So let's say I toss this coin first time and first time I saw head. Now what? Well, my estimator would say the bias of this coin is one in favor of head. That does not make sense because I have just got one observation. Since no matter what the bias is my estimated bias would be one or zero, independent of the coin's actual chances.

And this is clearly wrong. And it's clearly wrong in the sense that I have got so few observations so I need to correct for them. Okay? Again, in the case of restaurants, each restaurant has a different number of ratings. So simple average is definitely not the best estimator.

It needs correction, correction due to a limited number of observations. Just the way estimating the bias of a coin needs a correction. What should we do? This is where another principle of probability comes to our rescue. It is known as the central limit theorem. It states that the error in the average estimator of the bias compared to the true bias behaves in a very specific manner.

To be precise, let's consider the normalized error. That is, estimated score minus the true score divided by the square root of the number of observations. This normalized estimator or error has a distribution called Gaussian or normal distribution with mean 0. This is named after a famous German mathematician named Carl Friedrich Gauss.

Gauss lived from 1777 to 1885. He's also referred to as Princeps mathematicorum. In Latin it means one of the greatest mathematicians since antiquity. Now here's how Gaussian distribution looks like. The variation in the estimation or the amount of error varies as true score x (1- true score). Therefore, if true score is 0 or 1.

That is, truly always tails or heads, that is, always disliked or liked. Then error is 0. Totally makes sense. Effectively in that case, one coin toss is enough. On the other hand, if the true score is close to half, then error is going to be large. The problem is, we do not know a priori which is the situation.

So we have to assume the worst possibility. That is, error is the largest. In that case, the key takeaway from this mental experiment is that the error behaves like 0.5 / square root of # of Observations and number of samples. Therefore the true score is most likely 1 larger than the estimated score- 0.5 / square root of # of samples.

And smaller than the estimated score + 0.5 / square root of # of samples. This suggests a robust approach to order or rank or rate restaurants. One, assign scores to restaurants as average rating-constant / square root of # of ratings that the restaurant has received. Now here what is the constant c?

In the case of the coin it was 0.5. We shall discuss briefly how we choose c. Two, use this score to rank order the restaurants as before. This way, if there is a restaurant which is new and has been rated very, very highly by very, very few people.

Think of the owner and his friends. It will not be able to fool the system. On the other hand, if the restaurant is truly remarkable because its average is high based on a very large number of reviews. Then of course the correction would be very small, in which case average score would dominate.

Now coming to the choice of constant c. Well, in the case of Yelp example, choice of c = 2.5 is great because the ratings are between 1 to 5. All right, this is great. Now we have found a solution to combat the challenge of sparse data. And the solution is, again, very simple.

4.3.3 Recommendation Algorithm 101: Using Population Averages

You are thinking through this carefully, and playing a bit of devil's advocate at this stage. Is there any situation where the solution that we just found is not good? Can you be too conservative? To answer this, let us consider an example. Suppose a brilliant but lesser known chef opens up a new restaurant.

Everyone who goes to the restaurant raves about it. In a few weeks it gets four reviews, few genuine, four reviews on Yelp which are all five stars. In this case, double correction would suggest that the score of this restaurant should be, well, 5, which is the average. Minus correction, which is 2.5 / square root of number of samples, here it being 4.

So it turns out to be, if you do the calculation, 3.75. Well, this would mean that when we order restaurants by this adjusted score. This new restaurant is potentially unlikely to show up in even top-ten restaurants. Because there would be many veteran restaurants with average score of 4.

Will this prevent this restaurant from ever becoming a top restaurant on Yelp? No, not really. Since hopefully over time enough people will visit the restaurant. And then this would naturally bring it to the top because correction would become smaller and smaller. On the other hand, our conservative correction will prevent it to be discovered early on by various Yelp users.

And this would make it longer for this restaurant to become popular. So this suggests that our conscious conservative correction is preventing some kind of exploration among users early on. If we wanted to encourage some form of exploration then we should do optimistic correction rather than the conservative correction.

That is the adjusted score should be average rating plus, not minus, plus, constant / square root of # of reviews. Is that it? Are there other corrections that can benefit scores of restaurants at the population level? Well, for one, the freshness of ratings or reviews is important. This is especially important in the context of restaurants.

As time progresses, the restaurant can become better or worse, depending on many factors. Including changes in chef, management, or whatnot. Therefore, it may make sense to put more weight on recent restaurant ratings, compared to those further in the past. So how do we account for this? In computing average rating, we shall assign weights to the ratings.

You see, before, we assigned uniform weight to each review or rating. But now, we can assign weight to each review or rating depending upon its age. For example, if we think of half-life, remember half life the way we think of in nuclear physics? Let's think of a half-life of a restaurant as 3 months.

That is, this is time scale at which changes in restaurants become relevant. Then we can define normalized age of review as actual age of review / 3 months. So for example, if the restaurant had a rating 12 months back, then the normalized age would be 4. And then the weight of the review would be exponential of (-4).

Says that as age increases, the rate decreases. Now, the new weighted average score would be computed as follows. (weight review 1 x score review 1) + (weight review 2 x score review 2)... / weight of review 1 + weigh review 2... Okay? Again, remember previously weights were uniform, that is, equal to one.

Now we are gonna change that. And correction in this average score would be given by, as before, a constant divided by square root. But not square root of just number of samples. But a weighted sum of the reviews. That is, weight review 1 + weight review 2..., there's a small twist.

We will also divide this number by the maximum of the weights among all end reviews. Again, if you remember when weights of all reviews were equal in the earlier setting. This will be nothing but the square root of number of samples. In summary, in this section, we learned the following.

The average rating or score is an excellent proxy of the restaurant's true score. That is, we learned the algorithm of population averages. Now, due to a limited number of reviews or ratings in the context of restaurants, it needed an adjustment. Conservative adjustments suggest reducing rating depending upon the number of reviews.

Fewer reviews leading to more reduction. More reviews leading to smaller reduction. Optimistic adjustments suggest increasing rating depending upon the number of reviews. With fewer reviews leading to more increase. The freshness of ratings is very important to take into account. Weigh recent ratings more compared to older ratings. So this is the end of the section on how to use population averages.

In the next section, we are going to understand how to use population averages when data is in the form of comparisons. Thank you.

4.4.1 Recommendation Algorithm 101: Using Population Comparisons and Ranking

Today we are going to discuss our second recommendation algorithm of how to obtain ranking using population comparisons. In the last section, we discussed our first recommendation algorithm based on population averages. It was super easy. In the context of restaurants, we assigned each restaurant a score that is equal to the average value of all the ratings it has received and then used that to rank them.

We discussed various adjustments to it, but in a nut shell that was it. In many scenarios, we do not have access to ratings and even if we might have ratings, we might not want to use them as is. This section discusses such widely present scenario. When preference data is available in form of comparisons.

Consider your favorite sport, say American football. Well where the ball is kept in your hand during entire period rather than by foot. Or actual football, or soccer. Tennis, cricket, chess, or even baseball. In any of these sports, the ultimate question that we all as fans are trying to answer is, how can we rank teams or individual players based on the outcomes of games?

The games are always played between two teams, like the New England Patriots and New York Giants. Or two players, like Roger Federer and Andy Murray. The outcome of the games is usually one of the teams, or players, wins, and the other loses. This is precisely a comparison between two teams, or players.

For example, if Federer defeats Murray, that it is a comparison between Federer and Murray, in favor of Federer. Or if Patriots defeat the Giants, then it is a comparison between the Patriots and Giants, in favor of Patriots. Given all these comparisons or outcomes of games, we want to stitch them together across teams or players to eventually obtain a ranking or ordering between all of them.

That is what happens in any sports championship or tournament, such as US Open tennis tournament. Clearly this is like the situation where we wanted to rank all restaurants. But instead here, we want to rank them based on comparisons rather than ratings as before. Comparisons are not like ratings.

There is no easy way to take population averages of them. This requires some careful thinking. That is exactly what we will do in this section. Before we go down the part of figuring out how to rank teams or players based on outcome of games, it is worth pointing out that this view is very useful even in ranking restaurants.

When we have ratings for restaurants. Recall the restaurant scenario. Each restaurant was given ratings by different patrons who visited them and were kind enough to provide their views. It is just like re-watching movies and then rating the movies. Now the problem with rating is that it is phenomenally subjective.

You and I might like food in restaurant equally but I might be stingy in giving ratings and you might be very liberal. What that means is that rating of four that you give, is different from the rating of four I give. Or depending on the mood of the day, I might be more or less liberal in giving my rating.

In a sense, there is nothing absolute about ratings. To explain this important nuance, I would like to play a small game with you. Suppose I give you a blue color square and ask you, tell me the hexadecimal code of this. You'll wonder what is that? This is like me asking you to rate the restaurant or my orthopedic surgeon asking me during my visit, on a scale of 1 to 10, please rate your pain.

How do I know what nine means? Or for that matter, I'm not even sure what is the difference between eight and nine. And I wonder every time I'm asked this question, should I tell him a high number so I can get his attention? Well, a better situation is going to an optometrist who asked you, compare this scenario versus that scenario.

Which one provides you better vision? And through a sequence of such comparisons, we find the perfect eye power leading to somebody like me, able to see clearly and without any headache talking to you. This is like asking question. Which one of two different blue squares is more blue?

It's very easy to answer for you and me. And it is definitely not more absolute than asking for ratings. Back to setting up restaurants. If we are asked to compare two restaurants, when asked preference for West Bridge or Catalyst, it means the same to me and the same to you.

And therefore, we can use comparison data between restaurants to rank them as well. But this is the place where you say wait, Yelp has collected ratings on restaurants rather than comparisons. Now what to do? No worries. Ratings provide comparisons. Suppose I have rated West Bridge four-star and Catalyst three-star.

Then I have implicitly compared West Bridge and Catalyst, in favor of West Bridge. Okay so we are all comfortable with the fact that even in the restaurant setting, we have comparisons even though it was not obvious to start with.

4.4.2 Recommendation Algorithm 101: Using Population Comparisons and Ranking

Now that we have a bag of comparisons, between players, teams, restaurants, movies, or what not, we want to come up with rankings between all of them. This is a tricky question. This question has been a difficult puzzle for centuries. The essence of the difficulty was captured in the so-called Condorcet paradox, which is named after a French philosopher and political scientist named Marquis de Condorcet.

In late 1700s, around the time of French Revolution, he was one of the leading thinkers in understanding what is the best way to elect leaders in a democratic society. Naturally, in elections, we are trying to elect leaders by ranking, or ordering them collectively through our votes. In presidential elections in US, we cast vote for our favorite candidate only.

But suppose instead, we were allowed to cast votes in form of comparisons. That is, votes would consist of comparisons between pair of candidates, precisely the setting we have been discussing. And this is the setting that Condorcet was talking about. He considered the following hypothetical scenario. Let's say there are three options, A, B, and C.

We can think of these options as candidates, or restaurants, or sports teams. Now suppose we have three votes, vote one, A's preferred over B, vote two, B's preferred over C, vote three, C's preferred over A. If we look at only first two votes, it naturally suggests that A's preferred over B and B's preferred over C.

So that means the ranking should be A be first, B be second, C be third, but the last vote contradicts this. Similarly, if we take any two votes and try to order options, we will now perfect ordering, but the remaining third vote will contradict it. This is a paradox, because we can not order them in any meaningful way, and this is what makes ranking really hard.

At this stage, if you are wondering whether modern political science or the so-called social choice theory has anything to do with our world of ranking and recommendations, you are right on the money. There has been a very long intellectual history in social and political sciences, starting early-1900s. And lately, statistics computer science, and mathematics have been active in this topic due to various modern questions, such as the topic of our discussion today.

I would suggest an interested listener to read up a few notable works. One, A Law of Comparative Judgement by Louis Thurstone, published in 1927 in Psychology Review. Two, Social Choice and Individual Values by Kenneth Arrow published in 1951, and republished in 1963. Back to our Condorcet paradox, clearly it applies to our setting, and hence we need to resolve it.

So how to resolve it? Well, one way to resolve this, is to try to assign scores to each option. The scores in the setting, like above, will be equal for all options. That will be successful resolution of the paradox for the particular three vote scenario. And when it will make sense, the scores might be different for different options, leading to a global ranking.

Let's start with one such simple approach. To recall, we are given a collection of votes or opinions in form of comparisons. Given this, we want to assign scores to all options, and here's how we're gonna do it. The score of an option equal to the number of votes are comparison in which it is a winner, divided by the number of votes of comparison in which it is present, as either winner or loser.

Use this score to rank the options. The option with highest score is top ranked. For option A, it participates in two votes of which it wins in one, therefore it's score is 1/2. Similarly, a score of B and C will be half, and this resolves that paradox. Nobody gets a higher score than other.

4.4.3 Recommendation Algorithm 101: Using Population Comparisons and Ranking

Let us understand the scoring rule. For this, let us consider a scenario where we have only two options, A and B. That could be candidates in an election, restaurants, or sports teams. Imagine what you are excited about right now. I'm gonna think of A and B as sports teams.

Suppose A and B play ten games between them. Out of these ten games, A wins nine and B wins one. So what should the scores for A and B be? If you follow about simple rule that I described, it would suggest that A won 90% of the games and hence its score should be 0.9.

B won 10% of its games, so its score should be 0.1, simple. But what if they A had won all ten games. Should we be giving score of one to A and zero to B? Think about it. This doesn't make sense because implicitly it seemed to suggest that B has no chance of winning against A, ever.

It is true that A is pretty good compared to B. Self explanatory in A winning all the ten games. But that does not rule out possibility of B not winning once in future. It is worth reminding us of our favorite coin example. One where we think of outcomes of games between A and B is the outcomes of tossing a coin.

Every time A and B play a game. We toss a coin. If toss comes up head, A wins. If toss comes up tail, B wins. Then we're asking the question that if we have tossed coin ten times and it has turned up head all the time, does that mean that the bias of the coin is one?

No. Remember We can have a small chance of this happening even when bias of coin might be much smaller than one. And more so if bias of coin is close to one. For example, if bias of our coin was 0.99 then it will take, on average, 100 coin tosses to see even one tail.

Putting it other way, whether your coin has bias of 0.990 or 0.995, You would most likely see the first ten outcomes as heads in either case. And so no way you can differentiate between these two scenarios with limited data. And the reason why we should not assign one to A and zero to B when we see that A has won all ten games.

All right, so we know that we cannot do that. But then what shall we do? Well, like most simple questions in life, this question has a fascinating history too. Let me ask this question another way. What is the chance of Sun rising tomorrow? Let me try to explain this with the following mind experiment.

Suppose every day in the morning before Sun rises, I'm sitting and tossing a coin. The coin turns up head, we see sun rising. If coin turns up tail, we see sun not rising. For age of Earth which is roughly 1.6 trillion days, give or take, we have seen sun rising every day.

So we have seen outcome of coin being head 1.6 trillion times. Now what is the chance that Sun will not rise tomorrow? That is, can we estimate bias of the coin based on our observations thus far? Like setting off A and B, where A has won all games, we can not rule out a small chance of Sun not rising.

So the question is, how small is it? Pierre-Simon Laplace, a French mathematician and physicist asked this very same question in the 18th century. He suggests a very simple answer. Well, in one extreme case, we will see that sun will not rise tomorrow then why not we go with that as our observation and open the biased estimate?

That is we add a correction of plus 1 to both of our observations, that is, number of times sun has risen and number of times sun has not risen. Or number of times A has won and the number of times B has won. And use them to obtain the bias estimation.

In the case of our example, we will take the true outcomes of A's loss and wins and add plus one to each. So in this example, where we had A winning ten games and A losing zero games, you will define the score of A as 10 plus 1 which is corrected victories divided by 10 plus 2, which is corrected total number of games it has played.

That is, 11 by 12, rather than 1.0. And this will give the score of B similarly 1 by 12, that's it. Effectively each option sports team or restaurant is given score equal to the fraction of games it wins, with the added plus one correction to allow for error introduced due to small sample size as we just discussed.

This is a reasonable scoring rule. What is so tricky about it? It's like saying that a team score is equal to the proportion of the games it wins. Hm, sounds too simple isn't it? Because if so you would like your team to keep playing with the weakest team in the league and hence it keeps winning all the time.

Not good or a robust code. Defeating a player ranked number one should definitely count for more than defeating a player with thousands ranked multiple times. In some sense, we want to have a weighted scoring. You want to weigh each victory, depending upon whom you have defeated. And then average.

Great, a lot of fun, we have covered so much history while dealing with games, coins, and comparisons. Is there any hope of having more excitement in the same section today? Well, if you doubted that, you were wrong. We have a lot more to discuss before we end this section.

4.4.4 Recommendation Algorithm 101: Using Population Comparisons and Ranking

The big question is how do decide these weights. To answer this question, let us take another look at the same example of two options. We have A and B as our two options. They help play 10 games out of which A has won 9, B has won 1.

The character numbers are 9 plus 1 equals 10 in favor of A, 1 plus 1 equals to 2 in favor of B. The scores were, therefore, 10 by 12 in favor of A, 2 by 12 in favor of B. There's another way to get to the same answer.

Let us design a dynamical system to compute our scores. Think of a scoring machine if you like. We have two state A and B at each time we will be either be standing in state A or standing in state B. At each time, we decide to stay where we are or we decide to change our state at random.

The rule of this random movements are as follows. If you are at state A, then we stay at state A, with probability 10 by 12, or we go to the B, with probability 2 by 12. If we are at state B, on the other hand, then we stay at B with probability 2 by 12, and go to A with probability 10 by 12.

We run this experiment for a long time and observe the fraction of times we are in state A or fraction of times we are in state B. These fractions are scores of A and B of respectively. Effectively we have designed what is called a Markov Chain, over two states, A and B.

A transition rules between these states were designed so that the fraction of time spent in A and the fraction of time spent in state B, in the long term, but precisely equal to 10 by 12 and 2 by 12 respectively. If we carefully examine the transition rule it suggests that fraction of time spent in state A equals 10 over 12 times fractional time spent in A plus 10 over 12 times fractional time spent in B.

Since fractional time spent in A plus fractional time spent in B is equal to 1 by definition, we have that fractional time spent in A is equal to 10 by 12, voila. Now, this gives us a general approach to design weights we were looking for. In general, there are more than two options.

To explain this, and it starts with simple case of three options first. A, B, and C, how creative? We can create a graph of these three nodes, a node each corresponding to each options. As before, A and B have played 10 games, of which A has won 9, B has won 1.

For B and C, they have played 4 games, out of which each one of them have won 2 each. For A and C, they have played 6 games, of which C has won 4, A has won 2. As before, we can set pair wise corrected proportions for each of them.

A versus B would be 10 over 12 in favor of A. B versus C, 3 over 6 in favor of B. C versus A, 5 over 8 in favor of C. All right, so we've got our data. As before, we want to design a dynamical system, a scoring machine.

Where we can work between states A, B, and C. At each state, the rules of the transition to the next state are determined by which state we are standing in. And the other state to which we want to jump. This utilizes the outcomes of pairwise games. For example, if we are currently at state B, next time we will be at state A, proportional to 10 by 12.

And at C, proportional to 3 by 6. That is how often A defeated B and therefore, we will go from B to A proportion to that number. And that is how often C defeated B and that is why we go from B to C that often. To make sure that these numbers always add up to less than 1, we will multiply all of these numbers by a third.

This normalization number that is one-third here is obtained by looking at maximum number of neighbors, any node or any state has in this graph. In our case, all 3 nodes are connected, and hence, it's 3. Once we have this, we get the following effective transitions. When we are at B, in the next time, we go to A.

The chance 10 by 12 times one-third, which is 5 over 18. We go to C, one-half times one-third which is one-sixth. And then we remain at B rest of the time which turns out to be 10 over 18. The similar manner, we can design rules for A and C.

In particular, if we look at transitions from A to B and C to B the transition rate should turn out to be 1 over 18 and 1 over 6 respectively. This gives us a weighted formula. In the long run, the fraction of time spent in B is equal to 10 over 18 times fraction of time spend in B plus 1 over 18 times fraction of time spent in A plus 1 over 6 times fraction of time spent in C.

That is, rearranging this term, we obtain fraction of time spent in B equals fraction of time sent in A plus 3 times fraction of time spent in C all divided by 8. Similarly, looking at A and C, we get two more such equations that relates to fraction of time spent in each of these states.

The weights are coming from the relative wins and losses between the options or teams. These fractions will be the scores of options. Examining the equation for score of B again, score of B equals to, one-eighth times score of A plus three-eighth times score of C. This score naturally incorporates the fact that score of B is 3 by 8 times the score of C and 1 by 8 times the score of A.

The weights are coming from how often B has been relatively victorious over A and C. And if indeed score of C will be high, then score of B will be high, and vice versa. As we can see, when there are more than three options, the rules can be defined in a very, very similar manner.

The resulting ranking algorithm that we just learned is called Rank Centrality. Recapping the algorithm, it is computing scores as follows. We have a network of states. Each corresponds to possible options such as sports team or restaurants. A pair of nodes have connections between each other if they have been compared once or more.

At each time step, we are at one of these nodes in our machine. And we transition to any of the neighboring state next time but probability that is proportion to how often that neighbor has defeated us, the current state. So we are likely to go towards the winners more.

And less likely to go towards losers. And then we assign the score of each node as a fraction of time we spent in that particular node as part as dynamics in a long term. In the language of Markov chain, this fraction of time is called the steady state distribution of this design, the Markov chain.

Given this algorithm, a restaurant that is new, but has been compared very favorably to an old, well known Michelin three star restaurant might have a very high score and thus resulting into really meaningful scores. In summary, in this section, we discussed the following topic. In many scenarios, the preference data available is not necessarily with ratings, it is comparisons such as those in sports.

Comparisons are more absolute way to capture preferences compared to ratings. Comparisons can be easily derived from ratings. When preferences are comparisons, obtaining global ranking from them is not easy. Condorcet's paradox is a nice example that explains this difficulty. We discussed a simple fraction of wins as a scoring rule to decide ranking.

Naturally, it turned out to be too simple and it needs a weight adjustment. We used a dynamic system, a machine, Also known as Markov Chain in probability to design such an adjustment. The algorithm that we describe is known as the Rank Centrality. In the process, we also touched upon the questions of, what is the chance that sun will rise tomorrow?

And if it indeed does, we will see you in the next section and we'll discuss how to move beyond the simple algorithm that is just based on population view, and start making algorithms really personalized. Thank you.

4.2 Collaborative Filtering

4.5.1 Recommendations Algorithm 201: Personalization Using Collaborative Filtering Using Similar Users

Recall that our problem is to fill out a giant matrix using only a tiny fraction of its entries. In the case of the Netflix price data set, each row corresponds to a user and each column corresponds to a movie. Here is Frank, and here are the movies that Frank rated.

Here is the Dirty Dancing that Frank rated 5. In the previous video, showed us a method to rank items or movies from noisy observations of their ratings or scores. One of the crucial assumption that he made was that there was only one single score for each item and therefore one single ranking.

Assume, for example, that the 500,000 users in the Netflix dataset are all of the same type as Frank. So let's call them FRANK 1, FRANK 2, FRANK 3, etc. They would give the following ratings for the movie, Pulp Fiction, say. FRANK 1 would rate it a 4. FRANK 2 would also rate it a 4.

FRANK 3 would rate it a 5, FRANK 4 would rate it a 3, etc. So rather than thinking about all these Franks, we might as well think about the average Frank, who's rating is 4 + 4 + 5 + 3, etc., divided by the total number of Franks, which is 500,000.

This gives the average score for the Franks, say 4.1. So we could then fill out all the missing entries for the movie Pulp Fiction by entering 4.1 for all users. And this would be good if all users had essentially the same opinion of Pulp Fiction apart from small fluctuations.

This approach might be good enough for Yelp. After all, we're all comfortable with the idea that there's a universal scale of good food, and culinary guides, such as the Michelin guide or Zagat, are built on this very premise. More generally, we live in a society that's obsessed with rankings.

And the very notions of top 100 places to live or top ten party schools implicitly assume that every one has the same preferences in these matters. Of course, these notions are very subjective. And this is where personalization enters the picture. Believe me when I tell you that my preferences are very different from Frank's preferences.

I don't like Dirty Dancing, I'm more of a Jerry McGuire fan myself. But if anyone asks, I'm into action movies, I like my Bourne Trilogy. Now let's assume that Netflix has only two types of users. All of them a version of Frank, who loved Dirty Dancing, and hate The Bourne Identity.

And the other half are versions of me, who have exactly opposite movie tastes. If there are only action movies or chick flicks, then the full matrix would now look like this. Here, the matrix has been color coded for better visualization. A darker color means a higher rating. With the extra fluctuations that accounts for small variability among users and movies, it would look more like this.

The low ratings would be a bunch of numbers closer to 1, and the high ratings would be a bunch of numbers closer to 5. This picture is somewhat misleading, though. You get such a matrix when the first rows correspond to users similar to Frank and the first column corresponds to action movies.

In reality, the rows and columns are in a scrambled order and even in the simple case, two types of users, Frank and me, two types of movies, chick flick and action. The picture is much more complicated and looks more like this. This picture actually represents the matrix that we're trying to reconstruct, but remember that we don't observe most of the ratings.

They're only 1.2% of the entries of the Netflix matrix that are known. What we actually have to work with is this. In this matrix, white means that we do not observe the entry. From this mostly empty matrix, our goal is to recover the original checkerboard pattern. Good luck with that.

It turns out that a technique called collaborative filtering allows us to solve this problem using a basic machine learning method called nearest neighbors.

4.5.2 Recommendations Algorithm 201: Personalization Using Collaborative Filtering Using Similar Users

Recall that each row of our data matrix corresponds to one of the two types of users. Even if we observe only very few ratings per row, this should be similar for users of the same type. To see this, let's take a closer look at the similarity between rows

It should reflect the similarity between users' preferences. For each pair of rows, we can measure the similarity by looking at their inner product. This notion comes from linear algebra, and it's a single number that can be computed as follows. First, stack the rows on top of each other.

Here, we only show rows of size 10 rather then 17,000 for the sake of the demonstration. Then, we multiply each entry of the top row by the corresponding entry of the bottom row, and add these products. Of course, question mark times 4 is not well-defined, so we simply replace the question marks by the average rating, 3.

When we multiply each corresponding pair of numbers, the one on the top with the one on the bottom, we get this new sequence of 10 numbers. Finally, we add all these numbers to get 101. With it, the inner product between the top row and the bottom row is 101.

What does this number mean? Well, it's a matter of proximity between the rows, so the larger the inner product, the more similar the rows. Back to our Netflix example, a large inner product between two rows means that the corresponding users have similar preferences. In the toy example that we've just constructed, where there are two types of users, Frank and me, and two types of movies, chick flicks and action movies.

Let's see what these inner products look like. We could of course write a long list of inner products but there are a lot of them. Actually, this number is the number of distinct pair of rows, right? For each pair of distinct rows, we can compute a new inner product between them.

For n rows, this number of pairs is given by the formula n*n- 1/ 2. So, for n = 500,000 rows, as in the Netflix price data set, that number would be around 125 billion, which is about the number of stars in our galaxy. So, a list of all inner products would be pretty useless.

But we can get a good summary of this list using a histogram. In order to write an example, this is what a histogram looks like. We can clearly see that there are two different modes. This suggests that there are two types of values for these inner products, large and small.

When two rows come from the same user type, we get a large value. When they come from two different user types we get a small value for this inner product. This suggests the following picture. The blue dots correspond to users of the first type, ones that are into chick flicks like Frank.

And the red dots corresponds to users that are into action movies like me. Of course, this two dimensional picture is only a cartoon. In reality, this picture takes place in a 17,000 dimensional space. 17,000 is the lengths of each row in the Netflix dataset. Moreover, the clusters may overlap, like this.

Now that we have made this observation, we can devise a personalized recommendation system. The idea is simple and consists of using well known machine learning technique called nearest neighbors. Since users are clustered into two groups, even if they are not exactly the same, the ones corresponding to similar rows are likely to be users of the same type.

Assume that Frank is here. Then all the users that are in a certain radius are of the same type as Frank, chick flick fans, and their ratings may be averaged without risking to include the ratings of action movie fans. And that's personalization. Here, all recommendations will be personalized to each user type, either chick flick fan or action movie fan.

In other words, all the rows that have a large inner product with Frank's are likely to have preferences similar to Frank's. Once we have identified the set of users that are similar to Frank, we can apply the techniques that we saw in the previous video. Because for these users, there is only one single true rating for all movies.

This rating can be used to help with a single ranking. It is believed that the Netflix dataset has at most 15 different types of users. In a way, this is not such a large number, but it's certainly more than two. We'll see a straightforward extension to more than two groups, as part of the next video.

This video was about personalizing recommendations. We saw an example where the data contains at most two types of users. In this toy example, we saw how to identify users that are likely to belong to the same group by looking at inner products. Inner products between the rows of the data matrix are a measure of proximity between users.

Finally, we grouped users according to their proximity. This is the nearest neighbor technique and we computed an average rating within each homogeneous group. Effectively, we have personalized our recommendation to each group.

4.6 Recommendations Algorithm 202: Personalization Using Collaborative Filtering Using Similar Items

When you think of personalization, you think of user personalization. Each user should have his own recommendation, right? We know exactly how to do that. Group users according to similar preferences and fill the gaps based on users that are close to you. In the Netflix example, items were movies.

To measure similarity between users we compute inner products between two profiles of users, that is two rows in the matrix. The main limitation of this approach is that these rows are mostly empty. Each user has rated a very small fraction of movies. We end up multiplying a lot of true ratings by our artificial rating, three, that we use as a place holder for empty entries.

Recall this example. Here, about half of the entries were missing, the red entries, and that's where we put a three. In practice we color code black for a present entry and red for missing entry. The picture looks much more like this. It's a drop of black in an ocean of red.

This has pretty drastic consequences when we're taking inner products. Let's try to align two such rows on top of each other. As you can see, there's very few overlap between the black present entries. To measure the extent of this problem, we need a better understanding of how missing entries are distributed across the matrix.

How are these entries field and spread across the movies? Did some users rate more movies, or did they all rate pretty much the same number of movies? Were some movies rated much more than others? To answer these questions, let us look at some data. Let us start with the number of ratings per users.

As we can see, it has a really long tail. A few users rated up to 40,000 movies, but most of them rated a much smaller number of movies. This image is consistent with having each user rate each movie at random with a probability of 1.2% independently across users.

In technical terms, this looks like a Poisson distribution. How about the number of ratings per movies? This is essentially the same phenomenon. A few movies have received up to 250,000 ratings, but most of them have received very few ratings. The most rated movies include Miss Congeniality and Independence Day for example.

The impact of the second plot is much more drastic though. Indeed, if a few movies have a lot of ratings then they're probably popular, which in turn means that they are likely to be watched and then rated. Therefore, it makes sense to focus our efforts on these movies.

For these movies, we have a lot of ratings. And therefore, we can take the inner products between columns rather than rows. It has a significant advantage over taking in products between rows. There are many more collisions. Let's make a quick calculation. The proportion of entries that are filled for the movies Independence Day and Miss Congeniality are about 250,000/500,000, which is about one half.

Therefore, if we place the two columns next tot each other and compute their inner product, we an estimate the number of collisions as follows. We have 500,000 times one half for the first column and one half for the second column, which is 125,000 collisions. That's a lot, and the good news is that we'll get a large number for most of the popular movies.

In turn, this means that we'll have a much more accurate measure of similarity between popular movies. So rather than doing user personalization, we can do item personalization. And it's exactly the same picture as before. If Dirty Dancing gets ratings that are similar to Ghost, for example, then the missing entries for Ghost should be close to the entries for Dirty Dancing.

As we'll see in the case studies, this item based personalization is much more predictive than the user based ones. For exactly the reason that we mentioned before. The movies that really matter have a lot of ratings.

4.7.1 Recommendations Algorithm 203: Personalization Using Collaborative Filtering Using Similar Users and Items

So we've seen how to personalize recommendations by either grouping users according to their past ratings or by grouping items according to the past ratings they have received. I mentioned briefly that the item-based personalization is better in practice. However, there's something that's potentially even better. Combining both approaches into one recommendation algorithm that's both item and user based.

There might be something very specific about why Frank likes Dirty Dancing that comes from both the fact that he's Frank and the fact that movie is Dirty Dancing. It's the user item combination that really matters here. To see how we leverage this rough idea, let's go back to our geometry picture.

Remember when we talked about user based recommendations, we thought of users as being clusters in some space. In the case of user item based recommendation, this is also the kind of picture that we need to see. However, it does not take place in the same space. We need to find a space where we can see the pairs of user items.

This is the picture we had for user base recommendations. This idea is nice in principle but it looks great only in two dimensions, the x-axis and the y-axis. In reality, this picture takes place in a very high dimensional space. It has as many dimensions as there are movements, about 17,000.

There is one problem, it becomes difficult to measure similarities accurately in high dimensions. Recall that each user, for example, Frank can be seen as a row of length 17,000 which represents all Frank's potentials ratings for all the movies in the database. We have seen that the similarity between two users is measured by the inter-products between their rows.

Let us look at the most central user we can think of. The user that rates all movies a 3, the average rating. Next, assume that all other users are variations around the central user. Their ratings for each movie are as follows, with probability one-half, they rate the movie a 3, like the central user.

But with the probability of one-quarter, they rate the movie a 4, one point higher than the central user and with probability one-quarter, they rate the movie of 2, one point lower than the central user. We can represent each new row as the sum of a row of 3s for that central user and a row that contains only 0s, 1s and -1s.

Let us call the first row c for central and the second row d for difference. Now let us look at the inner product between two search rows, C plus D and C plus D prime, where D and D prime are two separate different rows taken independently at random.

Let's run some simulation to see the similarities between such users. This plot is a histogram of inner products between all pairs of 10,000 rows of the form D in the product with D prime for D and D prime random. We can see from this histogram that most of the inner products are between -200 and 200.

This range is not very large so inner products between users from the centrals cluster should stay close to the inner product between the central user and itself. This is (3 x 3) + (3 x 3) etc., 17,000 times which gives an inner product of 153,000. Therefore 200 is negligible compared to that large number.

Unfortunately, 200 is still a very large number, and it may be difficult to separate this cluster from other clusters. Indeed, we would like to think of all the users that are essentially the same as the central user up to this random variations and therefore as one cluster of user.

Visually, this cluster may be difficult to single out, because it overlaps with clusters of other users. For example, if we consider a user that rates all the movies 3, except for movies about dancing, that are rated a 4. If we assume that all the dance movies correspond to the first columns, the row corresponding to this user would look like this.

Now, consider the same type of variations around this user. This is our second cluster, let's call it the dance cluster. We can create two histograms. The first one that has only the pairwise similarities between rows from the central cluster, that is, variation around the central user and pairwise similarities between users in the central cluster and users in a dance cluster.

In our simulation, we assume that there are 100 dance movies. We can see that there's a huge overlap between the two clusters. In other words, given a user in the central cluster, there are many users in the dance cluster that appear to be more similar to the central user than users in the central cluster, this is a problem.

We could overcome this problem by only looking at ratings for movies about dance. In this case, the picture would look like this. The two clusters are very well separated. Moving forward, the main question is how can we automate the selection of dance movies without knowing it? Well this is exactly the idea of similarity between movies.

If they receive a similar rating, then chances are they are about similar topics, for example, dancing. User item based recommendation is simultaneously performing this clustering of movies and users. As it turns out, the problem is completely symmetric. I could have told exactly the same story by switching the role of user and items.

We have seen that because the problem is so high dimensional, it might be necessary to group or cluster users in movies simultaneously. This is the subject of user-based recommendations, our next video.

4.7.2 Recommendations Algorithm 203: Personalization Using Collaborative Filtering Using Similar Users and Items

Let us try to visualize what the whole matrix would look like if we were to order rows and columns in such a way that similar rows would be next to each other, and similar columns would be next to each other. Visually, this amounts to going from this picture to that one.

In this cartoon there are only two blocks, but in reality we may have more blocks and they also may overlap. Each block corresponds to a group of users and a group of movies that are homogeneous. For example, it could be that this group of user would tend to rate this group of movies a bit higher than the rest of the movies, or a bit lower.

It could also capture a bit more complicated pattern that is different from the rest of the matrix. The key mathematical observation is that a matrix with a block structure can be decomposed as the sum of simple matrices, one for each block. If these matrices are different enough, we can efficiently recover each of them using a numerical algorithm called singular value decomposition or SVD in short, even when the matrix is only very partially observed.

Let's see what this simple structure is. For that, assume for simplicity that our matrix has only one block and that it is located on the top left of the matrix. Here the red part is the pattern we are looking for, and the blue part is the rest. For example, the blue part corresponds to only ratings from the central clusters.

3 is with probability of one half, 4 is with probability of one quarter, and 2's also with probability one quarter. However, assume that the red block corresponds to bikers and dance movies. As we all know, bikers have a thing for dance movies and tend to rate them a bit higher.

Let's say that they rate them 4 with probability one half, 5 with probability with one quarter, and 3 also with probability of one quarter. So in average, one point higher than the central cluster. If we call B the blue matrix then we can actually represent the superposition of the blue and the red matrix.

As the sum B + U outer V, where U is a column of the same height as B, and V is a row of the same width as B. Specifically, U is a column with only 0s and 1s, where the 1s indicate the user is part of the bikers group.

And V is a row with only 0s and 1s where the 1s indicate the movie is about dancing. We say that U out of V is the outer product between column U and row V. We say U outer V. Unlike the inner product, the output of an outer product is a matrix rather than a single number.

It's easy to compute. The number of rows in the output matrix is the number of entries in U, and the number of columns in the output matrix is the number of entries in V. Next, the entry of the output matrix in the ith row and jth column is simply the product of the ith entry of U, Ui, and the jth entry of V, Vj.

In the case of U and V, there are only 0s and 1s as we described. We get exactly a matrix with only 0s entries, except for the top left corner that has only 1s in it. It is clear now that B + U outer V corresponds to the matrix that we have described.

Then we can add more blocks. B + U1 outer V1 + U2 outer V2 + U3 outer V3, etc. We can add one outer product for each block. It turns out that the vectors U and V don't have to be made only of 0s and 1s, but may represent more complicated patterns such as polarized tastes.

For example, if they take value 1 on one group and -1 on another group. To illustrate this example, consider two groups of Netflix users, which we can safely consider to be disjoints, bikers and teenage girls. Then consider two groups of movies, 80s dancing movies and pop concerts. Next, take the column vector U with 1s for the bikers and -1 for the teenage girls, and 0 for everyone else.

Take the row vector with 1 for the 80s dancing movies, -1 for pop concerts, and 0s for all other movies. What does the outer product, U outer V look like? Right, it will have 1s for the pairs, biker 80s dance movies. And 1s for the pairs, teenage girls, pop concerts.

It will also have -1s for the pairs, bikers, pop concerts and teenage girls, 80s dance movies. This indicates that while bikers tend to like 80s dance movies more than the average Netflix customer, they tend to dislike more pop concerts than the average Netflix customer. The opposite is true for teenage girls.

Of course, more complicated patterns arise in practice. Given that our matrix of interest has a decomposition of the form, B + U1 outer V1 + U2 outer V2 + U3 outer V3, etc. It turns out that it's not too hard to find the pairs Uj, Vj using what is called the singular value decomposition, or SVD, as long as there are not too many such pairs.

Even if we only observe a small subset of the matrix. This is because the SVD is very robust to noise. In a way, each pair corresponds to a stereotypical rating pattern. If we allow too many such pairs, we have no hopes to recover the underlying matrix. But in practice these matrices are nice.

It is believed that there are about 15 such pairs for the Netflix matrix. To be more precise, the Netflix matrix has probably many more pairs, but it is well approximated by such a simple matrix. In the case studies, we will see how the singular value of the decomposition can tell us how to identify how many pairs are relevant for a given problem.

4.3 Personalized Recommendations

<u>4.8.1 Recommendations Algorithm 301: Personalization Using Comparisons, Rankings and User-Items</u>

Is personalization using comparison data. We have come quite far in learning about recommendation or personalization. Starting from the simplest personalization algorithm that utilized population level information. In the last few chunks, we have discussed various ways to develop highly personalized recommendation algorithm's. The algorithms we discuss they're primarily doing matrix completion.

I suggest you do check the part of this module on ranks centrality again, especially if you want to remind yourself how to calculate the chance of sun not rising tomorrow. Now coming to the question of interest for the day. We shall discuss this question again in the context of Netflix.

There are a collection of users and movies. Each user expresses preferences over movies in the form of pairwise comparisons. Imagine Netflix showing the user a pair of movies and asking user which of these two movie did she like more? Of course, Netflix already has such data, as a user might have rated few movies already.

And the rating of movies implicitly provides comparison between the rated movies. For example, if I rated movie Casablanca, 5, and One Flew Over the Cuckoo's Nest, 4. Then I'm suggesting that I look Casablanca over One Flew Over The Cuckoo's Nest. Okay, so for each user, we know that she has expressed preferences in form of comparisons between a few pair of movies.

From this data, for each user, we want to figure out the ranking over all the movies for which she has not provided any preferences or ranking. In the setting of matrix completion, we wanted to predict unknown ratings. Here, we want to predict the order or ranking between all the movies for which we have no preference data for each user.

So clearly this seems like a much taller order than just matrix completion. I will convince you that that is not really any different from matrix completion. It is almost like matrix completion with few twists and turns. First let us transform our data in matrix form. As before, in our matrix we have rows that correspond to users.

Each column does not correspond to movies anymore. Instead, each column corresponds to an ordered pair of movies. For example, consider an entry. Let us suppose it's row corresponds to me as a user. And it column corresponds to a pair of movies. Casablanca or One Flew Over the Cuckoo's Nest.

Therefore if there are N movies in total, in principle we have N multiplied by N-1 divided by 2 columns. For N equals to 3, this will turn out to be 3. For N equals to 10, this will be 45 and so on. Now each entry in the matrix is either plus 1 or minus 1.

Of course, if it is missing, then we do not know what it is. But it will always be, either plus one or minus one. For example, if entry for row corresponding to me as a user, and column corresponding to Casablanca or One Flew Over The Cuckoo's Nest is equal to plus one.

Then it means that, I like Casablanca over One Flew Over The Cuckoo's Nest. And -1 means that I like One Flew Over the Cuckoo's Nest over Casablanca. Therefore for each user we will have a row with some +1s and some -1 entries. The remainder of the row will be blanks, there is unknown values.

Ideally we're going to predict +1 or -1 for each of the blank entry. So that it leads to ordering of movies for which user has not expressed preferences. For example, if for movie Iron Man and Superman, I have not provided preferences. That is the entry corresponding to column, Ironman compared to Superman is unknown, w want to predict it to be plus 1 or minus 1.

As we shall see, we may end up predicting it to be a number between plus 1 and minus 1 rather than a number equals to plus 1, or minus 1. And from this you would like to produce ranking over all movies with unknown preferences. Again in the example above, if there were only two movies, say Ironman and Superman, with unknown preferences we're really trying to figure out whether Ironman is preferred over Superman or the other way around?

And this is easy because we can look at the predicted value which will be between minus 1 and 1. If it is greater than 0 we shall say Superman is preferred over Iron Man. If it's less than 0 then we shall say Iron Man is preferred over Superman and that's it.

The question is what happens when we have more than two movies. In a nutshell, we have two challenges. One, how to predict this value between minus 1 and 1, for each pair of movies. Two, given these predicted values, how to compute ordering, or ranking, between all the movies with unknown preferences.

Short for one, we will use the matrix completion algorithm like collaborative filtering. And for two, we will use rank centrality. Nice, right? No extra work today. We will use what we have already learned. I love when this happens, and today is that day.

4.8.2 Recommendations Algorithm 301: Personalization Using Comparisons, Rankings and User-Items

Okay, so let me start by convincing you why we can use matrix completion algorithm like collaborative filtering. Note that after the transformation we did with our data, we have a matrix with unknown entries being plus one or minus one. The unknown entries that we want to predict is really matrix completion problem for that reason.

There is, recall, one such algorithm, user-user collaborative filtering. Suppose you and I are two users. We have provided preferences for a collection of movies. Some movies for which you have provided preference but I have not, some movies for which I have provided preference but you have not, and some movies for which you and I share preferences.

Let us suppose there are three movies for which both you and I have provided preferences. Let us say those three movies are A, B, and C. The corresponding pairs would be A compared with B, B compared with C, and C compared with A. These correspond to three different columns in our matrix.

For each of these columns, we have provided values of plus 1 or -1. I like A over B, I like B over C, and I like A over C. On the other hand, you might have liked A over B, C over B, and A over C. Looking at the overlap of these three columns, it seems that you and I agree on two out of these three entries.

After computing similarity using cosine of our vectors restricted to the overlap of these three columns, we get similarity of 1 over 3. Now suppose you have provided ranking of Iron Man preferred over Superman, but my ranking for Iron Man versus Superman is unknown. Then we can use similarity between you and me to transport your plus 1, that is, you liking Iron Man over Superman, to my setting with the similarity rate of one-third that we just computed.

Now suppose in addition, there is our common friend, say, Philippe, who has also provided preferences over A, B, and C, then his similarity to myself is -1/3. And now suppose Philippe does not like Iron Man over Superman, that is, he has given preference for Iron Man compared to Superman as -1, then we can use his -1 with weight of -1/3.

Putting all of these things together, we obtain prediction as follows. 1/3 for similarity between you and me, multiplied by plus 1 coming from you, plus -1/3 for similarity between Philippe and me, times -1 coming from Philippe's preferences. This whole thing divided by 2/3 comes out to be 1.

That is, our prediction is plus 1. That's nice. Effectively, you are like me, and you like Iron Man.

On the other hand, Philippe is not like me, and he did not like Iron Man. And hence, when we combine both of these weighted preferences together, they end up suggesting that Iron Man is preferred over Superman.

That is plus 1. But now suppose Philippe actually had liked Iron Man, and Philippe's similarity to me was -1 rather than -1/3. Then, the prediction would be, just quickly recalling the same calculation with different numbers, is 1/3 times plus 1 plus -1 times plus 1, and this whole thing divided by 4/3.

This after a simple calculation will turn out to be -1/2. This example provides intuition why we will see predicted values that are not necessarily plus 1 and -1, but values most likely in between. This example also explains how collaborative filtering like matrix completion algorithm can be directly used in our setting.

Now we will address the second challenge of obtaining ranking for movies using these predicted scores for movie pairs. Let us look at any two movie pairs, say Iron Man and Superman as before. The score of -1/2 between them suggests that Superman is preferred over Iron Man. One way to interpret this is in the form of random outcome of games.

To understand what -1/2 means, let us consider special values of -1, 0, and plus 1 first. -1 means that Iron Man always loses compared to Superman. Plus 1 means Iron Man always wins. And 0 means that Iron Man wins half of the time, Superman wins half of the time.

This suggests that if a predicted number between -1 and 1 is, say, x, then it can be thought of as tossing a coin with bias B, where B is equal to 1+x/2. When outcome of coin toss is head, which happens with probability B, Iron Man wins. When outcome is tail, Iron Man loses.

Therefore, we can convert each of the predicted value for a given user row by adding plus 1 to it and then dividing the resulting value by 2. This gives us probability of one of the movie winning against the other movie amongst the pair of the movies that the column corresponds to.

If it was minus half, then B equals to one-quarter, that is, Iron Man wins only 25% of the time against Superman. Now we are back to the setting where we have a collection of movies, and between a pair of these movies, we know what fraction of time one movie wins over another movie.

This is exactly the setting for which rank centrality was designed, to obtain global ranking. That means that we can now apply rank centrality over all movies and obtain a global ranking among them for each user separately. And this gives us personalized ranking for movies. In summary, we discussed how to obtain personalized ranking when preference data is available in form of comparisons rather than ratings.

There is nothing new about the algorithm. It is about putting two algorithms that we have learned earlier, matrix completion algorithm like collaborative filtering and rank centrality for obtaining global ranking between objects using pairwise comparison data. Next, we shall discuss other approaches for personalization beyond what we have already discussed.

Thank you.

<u>4.9.1 Recommendations Algorithm 401: Hidden Markov Model / Neural Nets, Bipartite Graph and Graphical Model</u>

Today we are going to discuss how to design recommendation systems using graphical models and neural networks. The primary algorithmic approach for recommendation or personalization that we have discussed so far has been based on the view of matrix completion. We view user preference data as a matrix. And the goal of the recommendation algorithm is to complete the matrix by filling missing entries in this matrix.

The view of matrix completion provides us a way to fill missing preference data. This view, however, has few limitations. First, the matrix completion view assumes that the relationship between preferences of users is somewhat simplistic. It could be explained by simple relationship that is behind collaborative filtering or matrix factorization algorithm.

In reality this relationship could be quite complicated. Second, the matrix completion view assumes that the user preferences are static. In reality user preferences evolve over time. For example, movies that may seem excellent choice now may not remain excellent choice a decade later. Today we shall discuss how to address these two challenges.

We will discuss an approach to address them using formalism of what is known as graphical models. Okay so, what are graphical models? Well, in a nut shell, graphical models provide succinct representation that can capture a generic dependency relationship between various objects of interest. This representation is of particular interest in statistics and machine learning because it is amendable to scalable computation.

Precisely what we shall discuss is how to use graphical model to address these two challenges. First, probabilistic graphical models will help us capture intricate relationships between user preferences. Second, graphical model, with Markovian structure, can capture the temporal aspects of user preferences in recommendation system. Let us start by discussing how probabilistic graphical models can help capture intricate relationship between user preferences.

For the purpose of explanation, we shall rely on our favorite Netflix example. We have users and movies. We shall assume that each user either likes a movie or dislikes a movie. For example, suppose we have four movies, Casablanca, Iron Man, Godfather, and Batman. Then we have four nodes in the graph, the edges between these nodes should capture the relationship between the movie preferences across all users.

Intuitively an edge between Casablanca and Godfather might suggest that there is some form of relationship between preferences of users for Casablanca and Godfather. For each user each node is assigned value plus one or minus one depending on whether the particular user likes that movie or does not like that movie.

As a collection, all users are defining a distribution over +1 and -1 value assigned to all nodes. It is this distribution that the graphical model captures using appropriate edge structure and associated parameters. And it is this distribution that is truly capturing the relationship between preferences of all users.

So now let us discuss, what are the parameters associated with the graph? To describe them, we shall consider a specific paramaterization for such a graphical model, also known as Pairwise Graphical Model. In this paramaterization, there is a distinct parameter associated with each node and edge of the graph.

In the four node graph example let us say theta one is the parameter associated with node one. And so on. There are a total of six pairs of edges and hence in principle there are six edge parameter associated with them. These parameters describe distribution of the four nodes taking plus one or minus one values as follows.

So let us say sigma one denotes the value associated with node one, could be plus one or minus one, and so on. Then, the probability of the four variables having this particular assignment is proportional to the following formula. You'll get positive formalized learning. Probability is proportional to exponential of summation of two terms.

The first term is the summation of four elements, each corresponding to one of the four nodes. The second term is the summation of six elements each corresponding to each of the edges. The graphical model does describe and cord the relationship between user preferences, or movies, through these node and edge parameters.

For example, if user population is such that all like all the movies, the parameterization where edge parameters are equal to zero and node parameters equal to infinity. Describes the corresponding graphical model. That is the graph has no edges. In practice, the parameter need to be learned from the observations.

The model capture relationship across all movies therefore at first glance it may appear that we will need to observe user's preferences over all movies to learn all these parameters. Can we learn parameters of the graphical model from sparse user preferences? The answer is somewhat surprisingly, yes. It suffices to observe very little data as long as across all users we have observed preferences between all pairs of movies.

We should be able to estimate the pairwise moments of all nodes. It means that we should be able to estimate the following. The fraction of users that like both movies, that like one movie but not the other one, and the fraction of users that dislike both movies. Amazing isn't it.

4.9.2 Recommendations Algorithm 401: Hidden Markov Model / Neural Nets, Bipartite Graph and Graphical Model

So how does one precisely estimate the parameters? One approach to estimate graphical model is called moment matching. We start by assuming some choice of parameters for each node and edge. We first compute the induced moment by the graphical model using given choice of parameters, and compare it with the empirically observed moment.

If there is a mismatch, it's likely change the node and edge parameters to reduce the mismatch. And repeat this process till we find a good match. That's it, very simple. It is important to compute the moments induced by a graphical model in a tractable manner. A popular approach to do so is the belief propagation algorithm.

I would suggest an interested listener to follow other parts of this course where graphical models are discussed in detail. So we have leaned a probabilistic graphical model of presentation of user preferences. How can we use it for the purpose of recommendation? Consider a user suppose she has liked only Casablanca, and that's all we know about her preferences.

Now we want to recommend her movies from the remaining movies. Using graphical model, we can compute probability of her liking any other movie, conditioned on the observation that she liked Casablanca. This is effectively assigning scores to each movie. We can use these scores to order movies and recommend them to her.

That's it. Computing the score, as described earlier, requires computing distribution of each node, subject to few observations from the entire graphical model. This is not a straightforward computational problem, a good computationally efficient algorithm for this is again the belief propagation, and this is exactly where graphical model representation comes to our rescue to allow for efficient computation.

Now we shall discuss the second challenge we had outlined in the beginning, how to capture the temporal relationship between user preferences using graphical models. If you like movie Goodfellas, then a good recommendation algorithm such as the probabilistic graphical model. A bow may suggest that you will like Godfather Part I, Part II, and Part III.

The question is, which one of these should be recommended first? As it is popularly believed Godfather Part II was the best among all three parts, and I think I believe that too. And if this was captured by user preferences, then our system might end up recommending Part II as the movie to watch for a user who has not yet watched Godfather Part I.

And this does not make sense, because If you have seen Godfather trilogy then you know that watching them in order is crucial. To make such a distinction, the recommendation system has to take the temporal aspects into account. Most user who would have provided their preferences for Godfather movies would have watched part one first and then part two later.

Therefore, if system utilized the temporal aspect of the user preferences, then such a problem would be solved. And that is what we will do next. Precisely, we shall describe how to use a hidden Markov model to solve this problem. We know that the so-called recursive neural network can be similarly utilized for this purpose as well.

And it seems like companies like Spotify uses such an approach for recommending songs to its users. In hidden Markov model, there is a notion of time. For each time instance there is a hidden state and there's an observed state. The hidden state is assumed to take one of the finitely many values.

The value of hidden state at time t plus 1 depends on the value of the hidden state at time t. The probabilistic relationship between consecutive hidden state is homogenous across time. In this sense, the evolution of hidden state can be viewed as a Markov chain and this is why it is called the hidden Markov model.

Now hidden states are not observed, well that is the reason why they are called hidden. Each time depending on the value of hidden state an observation is made, in our movie example, an observation would correspond to the movie preference of a user. For example, suppose a user has expressed the first preference as she liked Goodfellas and the second preference that she disliked Iron Man.

Then the observed state at time one correspond to she liking Goodfellas. The observed state at time two corresponds to she disliking Iron Man. In a nutshell, each user's preferences over time provide assignment to observed state of the hidden Markov model. Different user preferences are of different length, and hence we have observation of a hidden Markov model of different length, depending upon each user.

The goal is to learn the probabilistic relationship describing the evolution of hidden states as well as the probabilistic relationship between the observed state and the hidden state at any given time. A popular approach to learn this relationship is known as the Baum-Welch algorithm. An interested listener will be able to find the precise algorithm description in other parts of the course.

Now we shall discuss how to use of such a hidden Markov model for recommendation to a given user. As before, suppose a user has expressed liking for Goodfellas and disliking for Iron Man.

Then we know that the observed state at time one is Goodfellas is liked, observed state at time two is Iron Man, disliked.

Given this, using the learned hidden Markov model, we can compute the likelihood of any given model and liking being observed as a state at time three. This provides score for each of the movie, for which user has not provided her preferences. You can use the score to provide recommendations.

If the data is rich enough and algorithms have done their job right, then in the example above, it may suggest higher score for Godfather Part I, rather than for Godfather Part II. Even though at population level Godfather Part II might be liked more. That is the recommendation system would have worked well by understanding the importance of order in which sequels should be watched.

In summary, recommendation system based on view of matrix completion have two major limitations. One, they do not capture complex user preference relationship. Two, they do not capture temporal aspects. Graphical model can help us address these limitations in computationally efficient manner. We can use pairwise probabilistic graphical model to capture more intricate relationship.

We can use hidden Markov model to capture the temporal aspects of user preferences. The use of neural network is very similar to the way we describe the use of graphical models today.

4.10 Recommendation Algorithm 402: Using Side-Information

So far we have identified users and items to their corresponding row or column respectively. For example, for us, Frank was only a partially filled row of movie ratings. Frank is so much more than this row, but what really matters is what the recommendation system knows about him.

Since we live in the era of big data, trust me, the system knows a lot. Given more data, it seems natural that one would just want to increase prediction accuracy, right? In the Netflix example, if we just had more user movie ratings then one would simply be able to build a recommendation in our rhythm for Netflix.

Using the same user item base recommendation techniques that we have develop previously. What if the extra data does not consist of rating but still contains relevant information? It should still be able to help, the main guiding principle to use such additional information is to refine your model. Indeed just like we started with a completely homogenous and admittedly dull model that assumed that every movie pair was given the same rating.

All the way to the more refined user item based recommendation algorithm, we can refine this model even further. The idea, is to use side information to first group users in items into more homogeneous groups and then work on each group separately. To illustrate this point, consider the following diagram, we start with a large pool of users and a large pool of movies.

Then we use that information to group or cluster homogeneous users into say, three groups and homogeneous movies into say, four groups. Then on each pair of groups, we perform a user item based recommendation as we've learned how to do, that using only past rating. So what kind of information is available?

One thing for sure is that it should be relevant to our clustering purposes. Which is to say that it should have enough information to discriminate users and items according to preferences, at least partially. In this sense, not all data is created equal and it makes sense to keep in mind that when deciding which data to collect for this purpose.

In practice however, we collect as much data as possible and apply a variable selection, or dimension reduction technique to keep the relevant information. Take for example the Amazon recommendation system, what does it know about Frank? Well, it knows not only products that he purchased, but also what product he's looked at.

In particular, the dance related items, for how long he's read reviews, how much money he spends on weekends, etc. Overall, by now, Amazon has probably figured out that Frank is into dancing, what his income is, and whether he's a compulsive buyer. Frank is probably not the only

Amazon customer with this feature and Amazon may be able to identify users that are similar to Frank with respect to his features.

The same is true for items, Amazon knows a lot about items beyond their ratings, price, category, weight, color, etc. This is the information used to cluster items, let us take a closer look at how clustering can be done. There are many clustering techniques available out there, but all of them start from one simple idea, represent the objects to be clustered by a vector.

This is just an ordered list of numbers and then measure the similarity using inner products. We've done this in this very module when the vectors were the rows, or columns, of ratings completing with the average rating three when entries were missing. When we have sight information, we can create a new vector.

For example, we can summarize the sight information about Frank in the vector F = 1,0,1,24,152, etc. Which means for example that Frank had looked at gardening equipment the first number one has not looked at cooking books. The number zero has looked at movies, the second number one has spend 24 minutes browsing through dense equipment.

Spent $152 on this last purchase etc. Amazon may even have convinced Frank to subscribe to their credit cards and even know his buying pattern outside of the Amazon website. At the end of the day, the vector F that contains the same information that Amazon has collected about Frank is gigantic.

With hundreds of thousands of entries and some are more relevant than others. Let's say that one of these entries for example is that Frank has purchased a flight to Vegas with his Amazon credit card. Is this relevant or not? We could sit here and debate about this question from a socioeconomic point of view but we might as well let the data speak.

There are many ways to select variables in a data driven fashion, when the ultimate goal to get a better clustering. Some of them are covered in other modules of this course but to name a few we could use spectral clustering. Where the goal is to find a projection of the data before clustering.

Sparse methods where the goal is simply keep only the most relevant side information. Locality-sensitive hashing which is part of the broader family of approximate nearest neighbors. Most of these methods are coded in STEM software, but a few tweaks may be needed to adapt them to your specific purposes.

To conclude, let us note that the clustering described above, is a bit of a blunt tool, and often needs arbitrary decision. What if we have a continuum of users? How do we decide when to stop putting users in the first cluster, and start putting them in the second one?

Rather than making this adhoc choice, we can simply weigh users. We put more weights on users that are more similar to a given user. There are many ways to incorporate weights into recommendation systems, but here is a rather simple one to fix ideas. Recall that in absence of side information we computed similarity between users by looking at inner products between their rows of ratings.

And that we can do the same with their vectors of side information perhaps after variable selection. This gives us two Inner Products that we call IPR for Inner Product between Ratings and IPS for Inner Product between Side information. We can now combine the two by looking at a new hybrid similarity measure defined by IPR plus t times IPS.

Here, the parameter t is positive and represents how much weight we want to put on the side information. For small t, the similarity measure essentially disregards side information and for larger and larger t, it takes more and more side information into account. Choosing t is more of an art than science, the best choice of t will depend on how good your side information is and should be done in light of data.

Interestingly, there is an other interpretation of this hybrid similarity measure. This is the measure we get if for each user we concatenate the row of ratings with the vector of side information where each coordinate is multiplied by the factor square root of t. It is easy to see that when we take the inner product between the two longer rows, we recover the hybrid similarity measure.

We can also use different values of t for different users, and choose this value from inside information itself, we can use more complicated combinations. The hybrid measure that we've just defined is set to be linear, it is a linear combination of IPR and IPS. We could instead use nonlinear functions and this is what we do when we cluster first and then recommend.

In this case the rule to compute a hybrid similarity measure is as follows. If IPS is larger than a threshold, then use IPR as a measure of similarity. If IPS is smaller than a threshold, then use 0 as a measure of similarity. The role of t here is replaced by the threshold, which has to be chosen in a data driven way too.

To summarize, we have seen that recommendation systems often have site information about users and items available. This information can be used to refine our model that has removed some homogeneity assumptions. Practically this is done by incorporating side information when computing similarity between users or between items.

4.11 Recommendation Algorithm 501: 20 Questions and Active Learning

Netflix or Amazon are both web-based services and this is why they are able to collect vast amounts of data with minimal efforts. However our algorithms themselves did not exploit any aspect of being online. Indeed, even if Netflix suddenly lost Internet access and therefore connection with their users, they can still run the user item base recommendation system and predict missing ratings accurately.

What ability do we gain by being online as opposed to offline? Well, we can interact with the users. Get the information we need to move on to our next goal. Consider the following silly example. Let's say that Netflix has just bought the rights to add the movie, Moneyball, to their catalogue.

But they are not sure what to make of the side information that comes with it. Is it a sports movie? A finance movie? A teen movie? It has Jonah Hill in it. A chick flick? It has Brad Pitt in it. Let's say that it is difficult to automatically predict who is going to like this movie.

What Netflix could do is to recommend it to key users and adjust recommendations according to their feedback in real time. This is called active learning. This is in contrast to passive learning, where Netflix would select a large group of users and wait for everyone's feedback before making adjustments.

The advantage of active learning over passive learning has been demonstrated empirically and mathematically. Active is always better and is implemented in many aspects of our lives. Perhaps one of the first places where it was used was in clinical trials that are now run in phases. If a drug performs well on the first patients.

It will be used more aggressively on subsequent ones. But if it has adverse effects, the dosage may be adjusted in future phases or the trial may be cancelled altogether. The idea behind active learning is quite simple. Collect information as often a possible, not only to make future decisions, but also to guide future information collection.

Web interfaces are a wonderful tool to perform exactly that. Perhaps the most primitive form of active learning is the 20 questions game, where the person chooses a subject, say an actor, and a player must guess in 20 questions who this person is. If you have played this game, you know that you should split the population you have in mind by about half at every question.

For example, let's say the first question may be, is this person a man or woman? Or is this person dead or alive? Etc. As you are getting further down the game, you refine your questions. Let's say that you have used your first questions to establish that we are talking about a dead male actor, who plays in Westerns.

You may want to ask whether he was involved in politics. We train at this game from a young age by playing, Guess Who, for example. And there are websites that are freakishly good at this game actually. So how are computers so good at this game? Well, they know how to split the candidates into two halves very accurately.

This is called the bisection method. Why is it working so well? Let's say you are playing this game and start with a population of N candidates. After one good question, you are left with N/2 plausible candidates. After another good question you are left with N/4 plausible candidates, etc.

After K questions, you are left with N/2K plausible candidates. To be sure of your guess, you need to have isolated at most 1 subject. In other words you need to ask K questions such that N/2K is at most, 1. Solving for K by taking logarithms, this translating into K larger than log N where the logarithm is in base 2.

For example, if you started with 1 million candidates, you actually need 20 questions to be sure a win. The main difficulty of the game is of course the big questions that split the remaining candidates exactly in half, but computers are good at this. Let us go back to a more serious game, Tinder.

There're about 50 million regular users on Tinder. So in principle, your soul mate could be identified in 26 questions. However, the success of Tinder is based on precisely asking a very simple question, yes or no. Say, hot or not. Note that the question is not, is your soul mate's body mass index above or below 20.

This would be boring, and Tinder would lose customers. Finding your soul mate based on yes no questions seems like an impossible task. It's like having a three year old play the 20 question game. Is it Nemo? Is it Grandpa? Is it Mittens? Etc. It is the complete opposite of the bisection method.

Instead of splitting the population of size N into two parts of size roughly N/2, it splits it into two parts of size, N -1 and and 1, respectively. Unless we're very lucky, it's clear that we need N questions rather than log N questions in this case. Yet, Tinder does it and others also do it.

Kittenwar.com is a website that ranks the cutest kittens by only asking which is nicest out of two photos. The Beer Mapper is another such example. It's an app that's trying to find what kind of beer you like by only asking which beer you prefer out of two choices.

For the Beer Mapper algorithm, the goal is to find two beers, say beer A and beer B, such that by knowing if a user prefer beer A to beer B, then one gets a lot of information about the beer preference of that user. Let's say that the goal is to identify the user's favorite beer.

Then in the spirit of the bisection method, the goal is to eliminate half of the candidate beers after each comparison. How can we do this? Well, the Beer Mapper starts from a geometric representation of beers that is based on ratings and descriptions of beers that were written by beer experts on the website ratebeer.com.

It looks like this. Beer Mapper picks two beers that split the landscape of beers into two halves. Say for example the user prefers the beer on the left, so we know that the user's favorite beer is in this region. Then we pick two more beers from that new region and repeat until we have narrowed our candidate beers to one beer, the user's favorite beer.

Using slightly more complicated techniques, we can find a user's ranking of beers rather than a single beer. Here, the goal is to place the user in this landscape and declare the closest beer, his favorite beer, the second closest, his second favorite, etc. Placing this user can be done using a method similar to the bisection method that we've just described.

The bisection method can go a long way when we're searching for a needle in a haystack. A soul mate on Tinder, or a beer in an app. In this case, it is good to keep in mind that active learning can be very useful. In particular, it can be used to establish preferences in the context of personalized ranking.

Case Study

4.12 Building Recommendations Systems

Today, our goal is to discuss how one might go about building an actual recommendation system. A recommendation system in production environments such as that in Amazon, Netflix, or YouTube is a lot more than what we have discussed thus far in this module. To begin with, recommendation system is a service that is integrated as part of an interactive data driven system that operates in real time and at scale.

For example, in the context of YouTube, the primary purpose of the system is to deliver different media content to the user. The rule of the recommendation system is to identify which media content get displayed on the user's interactive interface every time the user performs an action. Now any such real time interactive data human system has three key function components that strongly interact with each other.

One, sensing platform. Two ,storage and computation infrastructure. Three, intelligence processing algorithms. The sensing platform is the one that provides ability to interact with the end user or customer in real time in terms of both collecting data as well as delivering appropriate data including recommendations. The storage and computation infrastructure makes it feasible to store the collected information about user behavior in real time, as well as perform meaningful computation in real time and at scale.

The intelligence processing algorithms utilize the data collected via sensing platforms and store it in the storage infrastructure. It performs appropriate data processing using clever algorithms such as those discussed in this module to extract meaningful information and make decisions such as recommendations. These decisions including recommendations, are delivered to the end user through sensing platform while storing or logging them in the storage infrastructure for future use.

The primary purpose of this module and the course at large has been explaining what are the appropriate algorithmic approaches for intelligence processing. The singular focus of this course has been the use of foundations from statistics and machine learning to arrive at these algorithms in a principled manner. We believe that this approach would enable the listener to be able to develop principled algorithms for intelligence processing, for the task faced by her.

Today, we shall briefly discuss few guidelines on how one might go about actually building such an interactive system, that can scale and operate in real time. Imagine that your system such as Amazon or Netflix or YouTube is serving tens of or hundreds of millions or even billions of customers.

An excellent example is that of Google search engine. As per some statistics, more than 2 million searches are performed every second. It is this web scale that we are thinking of designing the system for. For such a system, let us start by thinking about the sensing platform. In the modern environment, end users of the system end up interacting through effectively, some form of web interface.

The end user might be using a web browser or a mobile app, but in a nutshell, in any of these contexts, our system interfaces through what is known as API, that is Application Program Interface. The key here is to make sure that the user interface, be it mobile app or web interface, utilizes the correct API in the correct place.

For example, when a user clicks on a media content displayed, a request is sent to the API with appropriate message. In response, the API should provide streaming connection to the content of interest, and the user interface needs to change so as to display this incoming stream of content and potentially displaying additional information such as the recommendations received.

This aspect of system design falls under what is known as developing user interface or in short, UI. Currently, the use of JavaScript framework is very popular for designing highly interactive web interfaces. The creative aspect of designing UI is an extremely important topic and way beyond my artistic skills.

A good system designer may want to think very carefully about the design aspect. In a nutshell, from the perspective of building system, is all about providing the right set of APIs and defining them carefully. Now that we have APIs that interact with users by exchanging appropriate data between users and system, it is time to discuss how should the API support it's promised functionality to the user interface.

We wanna think of designing APIs such that potentially, hundreds of thousands requests per second are received. Usually, the APIs are hosted through an array of web servers sitting behind appropriate load balancing infrastructure. The web servers upon receiving request, end up performing various tasks. This primarily includes storing or logging the information received and obtaining information within the system which might be precomputed to send the response back to the user.

The logging or storing of data, requires a good infrastructure that can handle a high volume of data, and request reliably. Apache Kafka is an open source example of such an infrastructure, also known as Publish and Subscription, or Pub/Sub infrastructure. Data may be stored in a robust, distributed file system as a data log for both archival purposes as well as making structure data analysis.

Open source example of such a file system is the hadoop file system. To make structured create efficiently, such data maybe stored in a file format such as Arcade. These files can be queried in a standard database form using open source infrastructure like Spark SQL. For the purpose of computation, the data that is logged or some processed form of it is required to be accessed in a random access manner.

For this purpose, use of robust distributed databases are needed. An open source example of such a database is Apache Cassandra. Now moving to computation infrastructure. The computation or processing infrastructure is required to perform various computational tasks. On one hand, it may be needed to perform basic tasks in running the data pipeline.

On the other hand, it may be needed to perform complex algorithmic tasks to produce meaningful recommendations. In either case, to deal with the scale of data and hence of computation, it is essential to utilize distributed computation infrastructure. Architecturally, the Map Reduce is a popular solution. It is no surprise that engineers at Google have popularized such an architecture.

In fact, they have been the pioneers in developing such web scale infrastructure. The efficient realization of such architecture is provided by open sources like hadoop or, more recently, Spark. In a nutshell, this requires thinking of algorithm as composition of small, atomic tasks, each operating on small amounts of data.

Cleverly interleaved collection of such tasks, performed in a distributed and parallel environment, can lead to a desired solution. For example, let us consider implementing recommendation algorithms in such a framework. If we think of basic algorithm based on population averages in the context of restaurant recommendation, effectively, we want to compute averages rating for each restaurant.

One way to divide this into multiple small tasks that can be performed in parallel is to create a separate task for computing average rating for each restaurant separately. Each of these small tasks have to make query to get all ratings related to the restaurant of interest and then take average of these ratings, that's it.

Such computation can be done easily. After all, we are potentially adding a small collection of numbers. We have a distributed implementation of our first algorithm this way. Now let us think about a more complicated algorithm like item-item collaborative filtering, in this context. We can do the same, but it will require performing some pre-processing beforehand.

Recall that in the item-item collaborative filtering algorithm and apply to our restaurant setting. We estimate the missing rating of a given restaurant for a given user by taking the weighted average of ratings of other restaurants that are similar. Therefore, if we had a way to find top few similar restaurants for given restaurant very quickly, maybe through one query to some form of data base, then we can create tasks, one for each restaurant and run each of these tasks in a distributed parallel environment.

The big question is, how do we pre-process data so that we can find restaurants similar to a given restaurant of interest very quickly? As if we are making a query database, and we want to make sure that this pre-processing can be performed in a distributed parallel environment as well.

Well, the answer to this question lies in a very deep connection between computer science and mathematical analysis. More precisely, this boils down to creating a data structure known as Nearest Neighbor Index on all restaurants with respect to the similarity that collaborative filtering algorithm cares about. In effect, this requires mapping each restaurant to a point in Euclidian space, of very small dimension.

Remember our world is three dimensional in Euclidian space. This mapping respects the similarity structure induced by collaborative filtering algorithm, that is, for a given restaurant, the closest restaurant in the Euclidian space are those that are most similar to it. Once we have such mapping, searching for closest restaurant is very easy.

It's just as if I asked you to go and look for your closest neighbor, all you have to do is take a few steps in every direction and then find out the first one you meet, and then report back to me, that's it. Now one such class of pre-processing algorithms that achieve this magical mapping are known as Locality Sensitive Hashing and they're pretty efficient to implement.

Finally, let us think about implementing algorithm based on finding the singular value decomposition of matrix. Now at the core of such linear algebraic algorithms, one requires performing multiplication between matrices. The matrix could be very large. For example, think of a rating matrix with hundreds of millions to maybe billions of users and tens to hundreds of millions of product.

This matrix has 10 to the power 17 entries in principle. This is prohibitively large. Therefore, to perform matrix multiplication at such a scale, it requires clever division of data and computation in smaller tasks. Indeed, such a division can be performed and can be scaled using the Map Reduce like architecture.

We will not discuss the details here, but it can be an interesting exercise for a listener to think about how to multiply two extremely large matrices using very small memory footprint. Summarizing, to build a recommendation system usually is an integral part of an interactive real time system, requires carefully thinking about sensing platform, storage & computation infrastructure, and intelligence processing algorithms.

The primary purpose of this course at large has been, developing principled approach for processing algorithms. Today, our interest was to briefly discuss how these algorithms fit in the broader ecosystems of the overall data processing system. We briefly touched upon various popular open source solutions for different aspects of the system design.

I would strongly recommend an interested listener to look at various open sources available under the Apache software foundation. Finally, a parting remark. Today, we have computation infrastructure available for rent through various cloud platforms. Therefore, it may make sense to utilize them to get off the ground quickly, without worrying much about maintaining such systems.

And as time progresses, depending upon the scale and requirement, one may evolve to maintaining such infrastructure on one's own.

4.12.1 Case Study 1 – Movies

Thus far, we have spent time discussing variety of approaches for designing recommendation systems. Today, we shall put our learnings to practice. We shall do this by means of three case studies. We'll design a recommendation system for recommending movies, just like Netflix, design recommendation system for recommending songs, just like Pandora or Spotify.

And we shall design recommendations system for recommending products, just like Amazon or Wayfair. First case study, movie recommendations. We shall start by discussing recommendation system design for movies. For this case study, we shall utilize movie lens dataset. This dataset is freely available at grouplens.org. This dataset has been collected between 1995 to January 2016 at a website hosted by University of Minnesota as a part of their research project.

The dataset is a collection of readings for movies by users. The total number of movies are a little more than 11,000.. The total number of users are a little more than 30,000. In principle, each user can assign a rating between one to five, to each movie. Therefore, total number of ratings can be little more than 330 million.

In the dataset collected, we have less than 3% of these 330 million or so entries filled. It is an extremely sparse matrix. The goal of recommendation algorithm would be to fill in these 97% or so missing entries. Among the ratings reviewed about users, the distribution of readings is as follows.

Around 4% of entries are rating 1, 10% or so are rating 2, 30% or so are rating 3, 35% or so are rating 4, and rest are rating 5. That is roughly 20% are rating 5. It is worth noting that a good fraction of ratings are 3, 4, or 5.

Less than 15% of ratings are 0 or 1. However, there is enough variation in the distribution that a simple recommendation algorithm will not be able to get accurate predictions upon missing entry. Now, we shall apply various algorithms we have learned to this dataset and evaluate the performance. To be able to evaluate performance of an algorithm, we will need to know how well does an algorithm predict an unknown entry.

Now, if entry is unknown, then we will never know how good is the prediction. This is a standard dilemma all such evaluation tasks face. To overcome this, a simple but clever solution has been developed, which is known as cross validation. We dig the available data, hide a small fraction of it from the algorithm, and then ask the algorithm to predict the values of hidden data.

And then, compare performance of the algorithm by comparing it with the hidden ground tree data. In experiments to follow, we shall hide 20%, or one-fifth of the data, and use the remaining

80% of data for training. The choice of this 20/80 split is done at random. To evaluate the performance of the prediction, we shall utilize the metric, known as root mean squared error, or RMSE for short.

It is the square root of the average of the squared error between the predicted ratings and the true ratings. For example, suppose algorithms predicts ratings for position one and two as s1 and s2, or suppose the true ratings were r1 and r2, then the RMSE is computed as follows.

First, compute the summation of the squared difference between s1 and r1 as well as s2 and r2. Now, divide this by two and then take the square root of it. Now, where it is, you can see there are different recommendation algorithm on this dataset. To begin with, and to set the baseline, let us start with an algorithm that does not utilize any data, a random recommendation algorithm, which uses any of the five ratings at random and does a prediction.

Let's try to understand how well will this algorithm perform on our data. The algorithm will predict a rating to be 1 with 20% chance or probability .2. As we discussed earlier, as per rating distribution, the unknown rating is likely to be 1 with roughly 4% chance. That is, the error will be zero when predictive rating is one, which is with chance 4%.

Similarly, as for the rating distribution, the unknown rating is two, three, four, and five respectively with chances 10%, 30%, 25%, and 21%. And therefore, the expected or average squared error when prediction is one will be 0(.04) + 1(.1) + 4(.3) + 9(.35) + 16(.21). Sum it up all, you'll get 7.81.

Now, this is the squared error when prediction is one, which will happen with 20% chance. Similarly, when predictions are two three, four, and five, the resulting squared error when we did all the calculations just like what we did for second back, we'll get 3.63, 1.45, 1.27, and 3.09 just by third.

Therefore, the total average square error under random algorithm is 7.81 + 3.63 + 1.45 + 1.27 + 3.09. Okay, that's a lot. All divided by 5, which gives us 3.45. And the square root of 3.45 turns out to be 1.85, which is exactly the RMSE of the random algorithm.

The algorithm that uses no information about the data achieves RMSE of 1.85. This should serve as a baseline for performance of algorithms that we shall evaluate. We shall start with the first algorithm we had learned, the population average. In this case, for each movie, we shall compute average ratings, and then for a user for which the rating of this movie is unknown, we shall assign this average rating.

No personalization. The RMSE, when evaluated for this algorithm, turns out to be .983, much better than the random algorithm. So definitely a good start. When we look at the movies or shows with highest average rating, they turn out to be Lord of the Rings the Return of the King, Lost Season 1, Battlestar Galactica, Lord of the Rings, Two Towers, Arrested Development Season 2, and so on.

Clearly, these are not personalized recommendations as they are based on population averages. Now, moving on to personalized algorithm. We shall first look at collaborative filtering algorithms. Using the user-user collaborative filtering algorithm, we find that the RMSE is .85. This is down from .983 of the population averages, a major improvement due to personalization.

To see how it personalizes the movie recommendation, consider the following scenario. Suppose I have rated Love Actually five, Notting Hill four, Pretty Woman four, Sleepless in Seattle five, Forrest Gump four, and An affair to Remember four. Clearly, from these ratings, you might say I really like romantic movies.

Now, we can ask the collaborative featuring algorithm to produce personalized recommendations for me using the rest of the movie lens dataset. It will produce the top recommendation as follows. Titanic, Sex in the City Season 4, The Notebook, How to Lose a Guy in 10 Days, Steel Magnolias. Clearly, these seem to be a good fit for the profile of a romantic character.

We would want to quickly compare performance of other algorithms as we have discussed. The variation of collaborative filtering, which is the item-Item collaborative filtering, and apply to this dataset results into RMSE of .91. This is a poorer performance compared to user-user collaborative filtering, but definitely a step up from the population level algorithm.

Finally, we can also compare performance of matrix factorization-based algorithm. These are based on viewing the user movie rating matrix as having low rank structure, and the goal is to find this low rank structure based on the partially-filled matrix. A popular implementation for such a problem is known as the soft impute algorithm.

Using this algorithm for the dataset results into a little higher RMSE compared to the collaborative filtering algorithm. So in summary, for movie recommendation based on movie and dataset, the best algorithm seem to be user-user collaborative filtering. It seemed to achieve a very good level of personalization.

4.12.2 Case Study 2 – Songs

Now we're going to case study 2, song recommendations. Unlike movies, which are a lot fewer, songs are too many. Therefore, each song has large few data points compared to each movie and hence the recommendation system for song is likely to required different source of algorithm compare to the movies.

For the purpose of case study, we shall utilize the million song dataset that is available through a website hosted by Columbia University. We shall utilize a smaller subset for simplicity of exposition. This smaller set contains 10,000 songs. As before, there are users and each user either has listened to a song or not.

Equivalently, each user provides a rating of thumbs up or 1 for a song that they have listened, and nothing for the ones they have not listened. To value performance using class validation, RMSE seems an appropriate metric given that ratings are either 1 or 0. That is users have either listened the song or not.

Effectively, this is a problem of finding how well can a recommendation algorithm find top key songs for a given user. We shall utilize the metric of precision and recall. To define this notion, we shall suppose that recommendation algorithm provides R recommendation, say 20 recommendations. Precision is defined as the fraction of R recommendation that belong to the true top K choices of the user.

Recall is defined as the fraction of true top K choices of the user that are actually part of the recommendation. Let us take an example, suppose K =10 and R = 5. That is, user has top 10 choices, and recommendation algorithm provides 5 recommendations. Let 3 of the 5 recommendation songs are among the true top 10 list of the user.

Then the precision is equal to 3/5, or 0.6. Recall on the other hand is 3/10, that is 0.3. We shall compare population based algorithms precision and recall with the collaborative filtering algorithm. We find that item-item collaborative filtering performs the best. Specifically, when we compare the precision and the recall, we get the following summary.

For R = 20, there is 20 recommendations by algorithm. And K = 20 there is top 20 songs per user. We find that population average algorithm achieves recall and hence precision in this case of less than 0.05. On the other hand, item-item collaborative filtering algorithm achieves recall and again hence precision of roughly 0.25.

That's quite a bit of difference. Effectively, this suggests that as per item-item collaborative filtering, 5 out of 20 recommended songs are actually among top 20 songs of a user. In my mind that's pretty good performance. And this, compared to less than one song being part of your top 20 songs as per population level algorithm.

When we closely look at recommendations produced by the algorithm, we find the following quality behavior. Consider a user who likes songs like One and Only by Mariah Carey, Diamonds from Sierra Leone by Kanye West and Jay-Z, Not Big by Lily Allen and Love the Way You Love Me by the Pussy Cat Dolls.

The item-item collaborative filtering algorithm recommends In Person by The Pussycat Dolls. One More Sad Song by the All-American Rejects and Cheryl Tweedy by Lily Allen. This seem to make sense, well of course assuming that you know what this songs are

4.12.3 Case Study 3 – Products

Consider a very different recommendation setting. Recommending products to user while they are browsing an eCommerce website for purchasing. As discussed earlier, this is the problem that companies like Amazon face. We shall utilize Amazon review data set that can be obtained for non-commercial purposes from a website hosted by University of California, San Diego.

We considered the data set that has 22,000 or so electronic products. They are reviewed by 3,000 or so users. Each user provides rating between one to five. The rating metrics again is very sparse. Specifically less than .1% of data metrics is filled. We shall utilize again RMSE as the matrix to evaluate the performance of the algorithm in cross-validation setting.

The distribution of the known entries in the rating matrix are heavily skewed towards five. Specifically, little more than 50% have a rating five. Among the remaining, roughly 50% of entries, roughly 25% have a rating four, while one, two, and three have remaining 25%, roughly equally distributed. This is quite skewed, and therefore one would naturally expect that the population average algorithm will do reasonably well in terms of prediction.

Indeed, the RMSE of the population average turns out to be 1.27 which is not much worse compared to the best of the collaborative filtering algorithm based on item-item collaborative filtering which is RMSE of 1.17. So yes, personalization helps, but in this case, only by so much. Qualitatively, a user who has purchased a laptop is recommended a laptop pillowcase.

A user who has purchased a camera is recommended a camera lens. Totally make sense. In conclusion, the purpose of three case studies was to explain how to utilize algorithms we learned for the purpose of producing recommendations. In particular, depending upon the context, different algorithm end up working well for different scenarios.

For example, in the context of movie recommendation, we find that user-user collaborative filtering is the most effective. In contrast, in the context of song recommendation, since the number of songs are too many, the item-item collaborative filtering algorithm works much better. The matrix factorization-based algorithm work somewhat poorly for the movie recommendation system, and similar performance holds for other data sets.

This suggests that the model assumption of low rank behind the reading matrix may not be appropriate for real setting. When the rating distribution is very skewed, the population level algorithm performs comparatively with respect to the collaborative filtering algorithm, since there is limited heterogeneity in data, such as what we observed in product recommendation scenario.

Therefore, for an application at hand, you may wish to understand the characteristics of your data set, as well as utilize a collection of algorithms to decide which may perform the best. There is no one server bullet.

Wrap-Up

4.13 Wrap-Up: Parting Remarks, Conclusion and Challenges

We have now reached the end of this module on recommendation systems.

Yes, thank you for staying with us, but before we part our ways, let's see what we'll learn by reviewing the topics that were covered. First, scenarios. We discussed a wide range of scenarios, arising in the practice that can benefit from having a recommendation system.

Then we defined the basic problem. The formal description of the basic problem of recommendation. In particular, we discussed that effectively recommendation systems solve the problem of missing entries when viewing data as a matrix or more generally, a higher order transfer.

And we discussed algorithms. We discussed a collection of algorithms for addressing this question. We started the discussion with the simplest of these algorithms. It is based on population averages. And it is so simple it barely deserves the name algorithm. It was built on the basic assumption that the entire population preference is homogenous.

Then we moved on to a more refined, a personalized algorithm that is based on collaborative filtering. The model assumption behind collaborative filtering is that users or movies with similar preferences on observed entries are likely to also have similar preferences on the known entries.

Finally, we saw a series of algorithms that can be based on the notion of singular value decomposition from linear algebra. This was the most powerful algorithm that we described.

We also discussed scenarios where data does not come as an absolute score but in the form of comparisons or rankings. In these settings we've discussed how traditional recommendation algorithms can be naturally extended to deal with this kind of data.

We also discussed an interesting scenario where data can be collected actively from users in order to be able to extract information in more targeted, and therefore meaningful manner. We use the analogy with the 20 questions game and describe a principle approach to use this technique.

Each recommendation problem is different, and a good data scientist should take advantage of as much domain knowledge as possible.

This is why we also discuss various aspects of a good recommendation system such as the role of time in determining recommendations and the role of complex interaction networks, such as graphical models for designing recommendation algorithms.

In a nutshell, in this module, we went through of a variety of modeling and algorithmic approaches that can be useful to design a good recommendation system.

There are also the aspects that we did not discuss. For example, the various challenges associated to designing a robust system that can scale with large amounts of data, as well as operate in real time. Designing such a system is a serious endeavor that goes beyond data science.

Some of the key challenges are associated with the following aspects. Designing a user interface through web, app, or other to gather user preferences as well as other meaningful recommendations in real time.

Storing a massive draw of user data, so as to be able to utilize for performing computation at scale.

Designing a distributed, scalable, and high-performance architecture to implement a recommendation algorithm.

And achieving all this in a robust manner in geographically diverse scenario, while achieving really low latency. Addressing such challenges requires the adoption of an appropriate system architecture that is relevant to user interface design. Distributed file systems and distributed databases, data streaming architecture and scalable computation system.

These aspects are beyond the scope of this module. In the last segment of the module, we briefly touched upon various popular solutions that are adopted today. While it is far from an in-depth discussion, we hope that the aspects discussed in this module will get you help started.

Go recommend.

Module – 5: Networks and Graphical Models (Caroline Uhler and Guy Bresler)

5.1 Introduction

5.1.1 Introduction to Networks

Welcome to the module on networks. I am Caroline Uhler from the Department of Electrical Engineering and Computer Science and the Center for Statistics at IDSS. My research is centered on graphical models, causal inference, and applications of these statistical methods to genomics. And I also teach courses on these topics at MIT.

I'm Guy Bresler. I'm with the department of Electrical Engineering and Computer Science at MIT and the Center for Statistics at IDSS. Caroline and I will be your co-instructors for the module on networks. My research is on machine learning and network algorithms and I teach courses on these topics at MIT.

The goal of this module is to discuss a variety of network problems, both from the statistical and algorithmic perspectives.

There's a lot of talk about networks these days.

So what exactly is a network?

At a high level, a network is a collection of interacting agents or elements.

For example, the World Wide Web is a large network. You navigate in this network by surfing from one page to another page. Pages are connected via links. These links are what makes the web useful. The pages and isolation are not as interesting.

So networks are found everywhere, because fundamentally, most interesting phenomena are about interactions between basic entities.

I suppose this becomes especially clear when thinking about social networks. Many of us use Facebook, a network where you are connected to your friends, and they are connected to their friends, and so on. There's not much to do on Facebook if you don't have any friends.

In fact, not only the things you interact with in your daily life are networks. Even you, yourself are composed of networks. For example, each of our brains is one large network of neurons that are connected to each other. A single neuron is pretty useless, but amazing complexity emerges from the interactions between neurons.

Okay, so I'm convinced that networks are important. But sorts of questions can we hope to answer by studying networks?

We want to understand how network properties give rise to observed phenomena. This will allow us to make predications and to better design new networks. The structure, or shape of a network, can have a big effect on the whole system.

How so?

Thinking for example about news spreading in a social network. Certain people are very influential and effectively propagate information. Don't you remember the last time your tweet got retweeted by Justin Bieber and it went completely viral?

Well, Justin Bieber always retweets my tweets.

In order to quickly spread information in a network, it is important to understand which network structures increase or decrease efficiency.

This can then be used to design or improve networks, such as power grids, transportation networks, or to add important links to a website. Of course, the appropriate notion of efficiency depends on the setting.

Still, regarding network structure, there is a whole other set of questions regarding prediction. For instance, Facebook predicts who's friends with whom based on it's available data and it makes suggestions to each user based on these predictions.

Similarly, Google can use the network of links between websites to make predictions for websites that are of interest to you based on previously visited websites. This is of course related to the recommendation systems module.

While networks are interesting, in and of themselves, it is the processes that take place on networks that make them come to life. Conceptually, it is often useful to separate the network structure from the process on them. We can think about the fact that Justin Bieber has a lot of connections as a network structured property.

But then we can study the propagation of a tweet via retweets. Network structure helps determine the behavior of the processes that occur on them.

The propagation, or cascade of effects, is what makes a network behavior surprising and often seemingly unpredictable. These processes can exhibit quite intricate behavior but modeling the processes occurring on a network allows us to answer questions about them.

Okay, so let's make this concrete. Imagine we own a concert venue and we wanna advertise for an upcoming concert.

Like the upcoming Justin Bieber concert?

Exactly. So instead of just advertising on the radio or via ads on Google, I want to initiate an advertising campaign via social network. This can potentially be a lot more effective at lower costs because it's possible to target the right set of people. And also, people put emphasis on information obtained from people they trust.

One way to do this would be to give free tickets to a few selected people in exchange for them making a post about the upcoming concert.

Because we can't give away too many tickets, we have to choose very carefully to whom we give tickets. Let's imagine we're only giving away a single ticket. To whom should we give it?

We should choose the most influential person.

Okay, so let's give the free ticket to the person with the most friends.

Well, you can't just count how many friends a person has to determine how influential they are. It might be the case that most of these friends have very few friends themselves and news about the concert wouldn't propagate beyond the single hop. We want to choose someone who has many friends who are themselves influential.

So we want to study how to define a useful notion of influence and think about how to compute it for a given network. This all makes sense. Properties of the network help determine how the processes on it behave. There are a lot of networks out there and a lot of possible processes.

In order that we don't have to study this questions for each network separately, we want to understand what are the characteristic structures and important features of various types of networks

For example, what makes a social network different from the food web or a recommendation system. In this module, we will study different models for networks and processes on networks.

5.1.2 Definitions and Examples of Networks

Welcome back. In order to study networks, it will be useful to be precise about what we mean. Mathematically speaking, a network is an object called a graph. It consists of nodes also called vertices. And there are edges that connect the nodes. For example, in a social network each node is a person, and an edge between two people represent their friendship.

So let's discuss some more examples of networks. We've already mentioned the World Wide Web where each node is a page, and the edges correspond to hyperlinks. Similarly, the internet is a network. Here computers are the nodes, and the edges are network protocol interactions.

Power grids consist of generating stations, substations, and transmission lines connecting them. Another example of a physical network is the road network. Traffic flowing through the network is a network process.

Also most biological processes occur on networks. For example, a neural network consists of neurons that are connected by synapses. Neural activity, specifically the pattern of electoral signals or spikes, is a network process.

Another example is the gene regulatory network. Here the vertices are the genes, and the edges represent regulatory effects. In phylogenetic trees, the nodes are species and the edges represent direct evolutionary path. In the food web, the nodes again are species, but now the edges are drawn between predator and prey.

In a recommendation system, relationships between items or users can be represented by networks. Finally, another example is the citation network, with each node representing a research article and edges representing citations.

So after seeing all of these examples of networks, it's useful to try to categorize them. The World Wide Web for instance, is different from the internet in a basic way.

How so?

Well, the World Wide Web is a directed network, whereas the internet is an undirected network. Edges and directed networks have a directions. A website can link to another website that does not link back.

An edge in the internet represents a communication channel. This is typically bidirectional, so we use an undirected edge in this case. Food webs are directed networks, given by the predator, prey relationship. And similarly for citation networks, gene regulatory networks, and phylogenetic trees.

We don't see many gazelles eating lions, so directed edge sounds accurate. Another hint, internet, power grids, neural networks, road networks, and metabolic networks are mostly undirected. Although with that said, if there's a one-way street, you'd represent that with a directed edge. In Facebook, friendship is an undirected relationship. So you'd use an undirected graph. Twitter in contrast, allows you to follow people without them necessarily following you.

So it is best represented by a directed graph. Even within undirected networks, there is a distinction between the type of network representing internet or power grids, and neural networks or road networks.

How so?

While neural networks can have multiple connections between any two neurons, power grids have at most one edge between any two nodes, and they don't have any self loops. Such networks are known as simple networks.

Directed networks an also be categorized further based on whether or not they have cycles. Networks that don't have any cycles in the underlying graph are called trees. Examples are phylogenetic trees. If a directed network has no directed cycles, then it is called a Directed Acyclic Graph or DAG for short.

Note that one can have a cycle in the graph, just not a directed cycle.

Yeah, for example, food webs are usually DAGs, since there is an underlying hierarchy. Smaller animals usually don't eat larger animals. Also, citation networks are usually DAGs, since a paper typically only cites earlier papers.

The information and recommendation system can be represented by a network with a special structure known as a Bipartite Graph. In Netflix, we would have users on one side and movies on the other side. Each user would have an edge to the movies they had rated. A graph is called bipartite if the nodes can be partitioned into two sets, so that there are only edges between the two sets and not within a set.

So we've now introduced some of the most important classes of networks, and given many examples of networks we encounter on a daily basis.

We should also mention the different kinds of data that can be collected on a network. For example, in the World Wide Web, we can collect data on which page was accessed from which other page.

This data tells us about the presence of an edge. Alternatively, we can collect data just on the nodes. Say, how often how was a particular website visited? This also contains information about the underlying network.

So, lets wrap up by summarizing some of the questions we might be interested in by studying a network. What would you like to know about a social network?

For one, I would like to be able to identify people like Justin Bieber, in order to spread my tweets as efficiently as possible.

Great, so one question we will discuss in this module, is how to identify important nodes or edges in a network, and what makes a node or an edge influential.

More generally, we will talk about properties or characteristic of networks. We will learn about the kinds of structures that can be used to distinguish and identify different classes of networks.

So another question I'd loved to answer is, I had a cold last week and I want to know who I should blame for this? How do I infer how does a virus spread on a network? How can we infer how neighbors in a network influence each other? These are also questions that we will discuss in this module.

Also, you might not remember who exactly you had met with last week, but maybe we can recover the underlying network describing who interacted with whom, just from data on how the virus spread.

Similarly, we might be able to recover the links between websites, just from data on how often each website was visited. Learning the underlying network from data generated by process on the network, is another theme in this module.

Finally, we want to be able to make predictions either regarding the future, or for nodes that we did not observe.

Using the underlying network structure and data on a particular process. Was a certain person infected by the virus? Or how likely is she to catch it in the future? How likely is a new link between two websites going to be used? We do not know the answer, because the link does not yet exist, but a prediction will help us to decide if we should add it.

Whether we're interested in a spread of a virus, an idea, or a tweet, the same basic ideas will allow us to understand what's going on, and to make predictions about unobserved quantities.

5.2 Networks

5.2.1 Representing a Network

Welcome back. In order to answer questions about large networks, we need to be able to succinctly represent networks in a computer.

Why can't we just look at the network and figure out what we want from there?

Well, large networks usually look like a huge tangled mess.

Can you point to the most influential node in this network? Okay, so let's discuss how to represent the network in a computer and have the computer figure out which node is the most influential one. One way to represent this network is by an adjacency list. For each node, we list all the nodes it is connected to.

Here is an example for a small graph and ten nodes. With this representation you will easily be able to find the person with the largest number of friends. However, imagine that I am Facebook, and I would like to suggest friends to you. One puristic method is to suggest the person with whom you have many common friends.

In other words, I'm interested in pairs of nodes that have many paths of length too connecting them, but are not already connected by an edge. It's difficult to see how to do this using an adjacency list.

So let's be more clever about the representation of a network.

We can also represent a network by an adjacency matrix. An adjacency matrix is a two dimensional square array where the rows and columns are indexed by the nodes in the graph. The entries of the matrix are either zero or one, one in position one, two means that there is an edge pointing from node one to two.

In fact, we can even encode the strength of an edge in an adjacency matrix by replacing the ones in the matrix by other numbers, where for example an edge of strength ten could connect close friends where as an edge of strength one could represent acquaintances.

What happens if the network is undirected?

Well, an undirected network can be seen as a bi-directed network, meaning that when there is an edge pointing from node one to node two there is also an edge pointing back from node two to node one. This makes the adjacency matrix symmetric. Node I points to node J precisely when node J points to node I. So entry I J of the matrix is equal to entry J I.

So let's see how Facebook can use the adjacency matrix to efficiently suggest friends. Let's take two people in the network. Let's see if we should be friends on Facebook. In this small network we can easily count the number of shared friends that we have. But how do we do these using the adjacency matrix?

Well, this row is you and this row is me. In order to count the number of paths of length two between us in the network, we need to overlay these two rows and count the number of common once That's exactly the result you get when multiplying your row vector with my row vector.

Great. So doesn't this mean that by multiplying the matrix with itself, we can obtain the number of paths of length two between any two nodes?

Yes. Let's go through this carefully. When we multiply this by ten by ten adjacency matrix by itself, then entry IJ is just the sum over all K, of the product of entry AIK with entry AKJ, which is the number of paths of length two between nodes I and J.

So that's cool. So Facebook can just take the square of the adjacency matrix of the current network. Then you look at each row, and pull out the maximal entries that are not yet connected by an edge. These are good friend suggestions because of the many common friends.

But that means that I can do the same computation if I'm interested in finding friends of friends of friends. Or friends of friends of friends of friends. I just take the third power or the fourth power of the adjacency matrix.

Exactly. That's one of the nice features of working with adjacency matrices. But can't I use a similar argument to test if a directed network is a DAG?

How so?

Well a DAG is characterized by having no directed cycles. A directed cycle is a path from a node back to itself. So the number of cycles at length K in a DAG is just the sum of the diagonal entries of the Kth power of the adjacency matrix.

But that's known as the trace of the matrix. Hence a directed network is a DAG if, and only if, the trace of all powers of the adjacency matrix are zero.

But that doesn't seem easily checkable on a computer, since you need to take the powers until infinity. Wait a second. It's a well known mathematical fact the trace of all powers of a matrix are zero, if and only if all of its eigenvalues are zero.

And we know that we can easily compute eigenvalues using a computer. Even for very large matrices.

This means that we can check if a network is a DAG by just computing the eigenvalues of its adjacency matrix.

So with this we'll end the introduction to the module. We've discussed why networks are important, different classes of networks and the kinds of question you can expect to be able to answer after this module. Working with large networks is only possible with the help of a computer and we presented a convenient way to represent the network in a computer that allows to efficiently compute various quantities of interest.

5.2.2 Various Centrality measures of a Node

In the last few lectures, we have introduced networks, given many examples of networks, and shown how to represent a network using an adjacency matrix. The adjacency matrix representation allows us to use a computer to efficiently find various properties of the network. In the next three lectures, we will discuss different ways of quantifying and computing the importance of nodes, as well as other network properties.

When presented with a relatively small network, our eye is very good at picking up patterns, such as hubs or clusters. However, once a network reaches hundreds of nodes, it becomes difficult to visualize it. The network usually looks like one big hairball. So, we want to probe a network, find the important characteristics of the network.

Not by visualizing it, but from computations based on its adjacency matrix. In this lecture, we will discuss how to determine important or central nodes in a network. The importance, or centrality measure of a node, of course, depends on the application. So, let's return to the example of a social network like Facebook, and running an advertising campaign for the upcoming Justin Bieber concert via Facebook.

Assume we would like to give away one free ticket to the most influential person in the network. Hoping that this person will then motivate his or her friends to buy tickets and go to the concert as well, and these friends will motivate their friends, etc. This will result in a cascade of people buying tickets for the concert.

One strategy is to give away the ticket to the person in the social network that has the most friends. In a social network, each person is a node, and an undirected edge between nodes represents their friendship, so we would like to determine the node with the highest number of edges connected to it.

The degree of a node is defined as the number of edges that are adjacent to it. So, in mathematical terms, we would like to determine the highest degree node. How can we do this using the adjacency matrix? Well, the adjacency matrix contains a one in row i and column j, if person i is friends with person j, and a zero, otherwise.

So, if we want to know the number of friends of person i, we just add up all the entries in row i. And by computing each row sum, we obtain the degree of each node. Then we can give away the free ticket to the node with the highest degree.

However, as we've already discussed, we might not want to define the importance of a person just by the number of friends that this person has. If we want to achieve a cascade of people buying tickets for the upcoming concert, we should also take into account how important the friends of a particular person are, and their friends, etc.

So, if a person has many friends, but these friends have very few friends, then that person might be less influential than a person with fewer friends. But who's friends have many friends? So, how should we quantify this? Well, one way, is by the so called Eigenvector Centrality. This centrality measure gives each node a score that is proportional to the sum of the scores of all of its neighbors.

We've already seen that eigenvectors of the adjacency matrix can be used to determine if a directed graph is cyclic or acyclic. But what do eigenvectors have to do with the centrality or the importance of a node? Well, we would like to give each node a score that is proportional to the sum of the scores of all of its neighbors.

In order to do this, we need to know the scores of its neighbors. Put this way, it's not exactly clear how to start this process, since each score depends on the scores of its neighbors, which we don't know. So, let's just start things off by guessing a score for each node.

Since I don't have any background information, I will give each node a score one. I will store the list of scores of each node in the network as a vector, and I'll call this vector X. So, Facebook is a network with over a billion nodes. So, x is a vector of size one billion.

Now, we can compute each entry of x by replacing it by the sum of the centralities of all of its neighbors. For example, for the entry in x corresponding to guy, this means that we have to sum all the entries of x corresponding to his friends. Since the adjacency matrix has a 1 in position j of row i, if an only if person j is friends with person i, we obtain the new score of person i by multiplying row i with x.

This means that we can obtain the new score of everyone in the network, all at once, by multiplying the adjacency matrix with x. Okay. But we're not done yet. We need to repeat this process over and over again. In fact, one can show that this process stabilizes. Meaning that after many steps, let's say T steps, the vector XT, does not change compared to the previous score vector XT minus 1.

In addition, one can show that the final score vector XT is proportional to the eigenvector, corresponding to the largest eigenvalue of the adjacent matrix. So, that's how the eigenvector enters into the picture. Let's remind ourselves of the definition of an eigenvalue, and its corresponding eigenvector. So, given a matrix A, V is an eigenvector of A, if multiplying by A, doesn't change the direction of which V is pointing, it only scales the vector by lambda.

So, the eigenvalue lambda, and its corresponding eigenvector V, satisfy the equation A times V is equal to lambda times V. Since the score vector X is an eigenvector of the adjacency matrix, it satisfies lambda times X equal to A times X. Where lambda is the largest item value of A.

But this shows exactly that for example, my centrality is proportional to the sum of all the centralities of my friends. Exactly what we wanted. And since eigenvectors can be calculated efficiently, even for very large matrices, we can perform this computation even for very large social networks, like Facebook, for example.

So, using the eigenvector centrality measure, we might end up giving away the ticket to someone who doesn't have the most friends in the network, but who's friends themselves are very well connected. And whose friends are friends are also very well connected. So, we now know how to find important nodes in a social network where the edges are on directed.

But what about networks with directed edges? Take for example, the world wide web. Anyone can build a website that links to hundreds of other websites, so it's easy to get a high outdegree. But it is reasonable to assume that a page is important if it has many pages linking to it, or in other words, it has a high indegree.

Now the direction of the edges play a role, indegree matters, while outdegree doesn't. This can, in fact, can easily be taken care of when computing the eigenvector centrality. For directed graphs, the adjacency matrix is usually not symmetric, and this mean that it has left and right eigenvectors, which are in general distinct.

The right eigenvector captures the centrality from the in degree. Whereas, the left eigenvector captures it from the out degree. To determine the importance of a website, we would use the right eigenvector. Whereas, for example, in order to find a cell that is malfunctioning, and therefore, secreting a lot of protein signals to other cells, then we would use the left eigenvector.

So, if we're talking about websites, you might wonder, how does Google determine the list of suggested websites when you perform a search? Does it really use the right eigenvector of the adjacency matrix? In fact, it uses a centrality measure that is very similar to the eigenvector centrality. It's called Page Rank.

So, the eigenvector centrality has two basic issues that are addressed by page rank. First, consider a directed acyclic graph, so a graph with no directed cycles. Since all of the source nodes in a DAG have no incoming edges, their score is zero. Now, let's look at all the neighbors of the source nodes.

Since all of the nodes pointing into these nodes have score zero, their score is also zero. What this means is that eigenvector centrality of all nodes in a DAG are zero. And hence, does not contain any information at all. Now, this problem can be addressed pretty easily by giving each node a certain amount of centrality for free.

And that's exactly what a page rank does. Now, the second problem of the eigenvector centrality is that a node with a high score, that points to thousands of other nodes, gives all of these nodes a high score. However, when thinking about the world wide web, this is probably not reasonable.

Since the effect of being linked to from an important website is diluted by all the other websites that are also linked to. Nobody is going to try every possible link from a page with 1,000 links. So, page rank takes this into account by normalizing the scores that a website transfers to its neighbors by the number of outgoing edges.

So, a website will have high centrality. If important websites point to it, and those don't have too many outgoing links themselves. In this lecture, we've discussed how to determine important nodes in a social network, and how google sorts websites by importance. Eigenvector centrality works well for undirected networks, such as a social network, where the edges represent friendship.

Page rank works well for directed networks such as the world wide web. Page rank is a central part of Google's website ranking technology. And we saw today how it works. In the next lecture, we'll talk about others and neutrality measures.

5.2.3 other “Spatial” Centrality Measures

During the last lecture, we discussed two different centrality measures. Namely the eigenvector centrality and the PageRank. These centrality measures are useful for finding important nodes in order to advertise in social networks or for ranking websites in the World Wide Web. But they're less appropriate for other applications, so in this lecture we discussed other centrality measures as well as their applications.

Let's consider the problem of deciding in which city to build a new airport. You would first map out all airports that are close by and determine their region that the new airport would serve. Then it would be reasonable to choose the location of the new airport, so that it is most central in terms of average travel time to and from the airport to the cities in the region.

So, in this application, the vertices are the cities in the region, and the undirected edges carry edge weights, naming the travel time between two cities. So the adjacency matrix is symmetric, and the entry's of the matrix are zero, if there's no direct street connecting to said these without passing to us third city.

Otherwise the value and adjacency matrix is equal to the travel time. Although there are usually many possible ways to get from one city to another, it is reasonable to assume that people take the shortest route. This is also called the geodesic path. So the goal in this application is to find the city that has the shortest average geodesic distance to all other cities.

Thinking about airport location you might want to appropriately scale the distances by the size of the cities. This can be done but it's easier to avoid scaling things the first time around. So, how do we compute geodesic distances from the adjacency matrix? Well we've already discussed how to compute lengths of paths between any two nodes, for example, the adjacency matrix squared contains the travel times between any two cities that cross through a third city.

More generally, the adjacency matrix A to the power K contains the travel times between two cities on a path that goes through K minus one cities. So the geodesic distance of city I and city J is the minimum non-zero value of the I-J entry of entry of the adjacency matrix A to some power K and we range over all K's.

So this is another very nice example of how we can use the adjacency matrix to gain a lot of useful information. To end the discussion about where to build a new airport, let's denote by DIJ, the geodesic distance between city I and city J. So for each city, we can compute it's average geodesic distance to all other cities.

If we have N cities, than this is 1 divided by N-1 times the sum of geodesic distances to all other cities. And we will choose to build the airport in the city with the smallest average. This measure of importance of a node is called the closeness centrality and similarly to the eigenvector centrality discussed during the last lecture, there are various variance of the centrality measure.

To give another example, we all know how quickly rumors spread, and in particular, how quickly the message in a rumor gets distorted. If the goal is to reach as many people as possible with an accurate message, a reasonable strategy might be to start spreading the message from a person with a high closeness centrality who on average has a short geodesic distance to all other members of the community.

So we now discuss closeness centrality and different applications where one might be interested in this centrality measure. However, in other applications, we might not be interested in identifying the most centrally located node in terms of distances. We might instead want to identify possible bottlenecks of the network. So how should we define a bottleneck?

So, a bottleneck can be defined as a node whose removal from the network will most disrupt the information flow in the network. Equivalently a bottleneck is a node through which the largest quantity of information flows. We can identify such nodes by something called the betweenness centrality. The betweenness centrality makes various assumptions to approximate the amount of information that flows through each node.

It assumes that information flows at constant speed along the edges and that information takes the shortest path. Then the amount of information that passes through each node is proportional to the number of geodesic paths that go through that node. This measure is called the betweeness centrality. So we've already seen how to compute geodesic paths.

Now a very efficient algorithm to compute the shortest paths between all pairs of nodes is the so-called Floyd-Warshall algorithm. It's an example of dynamic programming. So, at it's heart, is the following idea. Let's start with just three nodes. So here it's easy to find the shortest paths between any two nodes.

Let's save these paths. Now we add the fourth node. So the shortest path between nodes one and two is either the shortest path that we already found in the small graph or there is a new shortest path that also uses node four. That path can be written as a sum of two paths.

Namely one from node one to node four, possibly using node three. And then one from node four to node two, possibly using node three. So you can see how you can find the shortest paths in larger and larger graphs by only knowing the shortest paths in smaller subgraphs.

In many applications it is important to be able to find nodes with high betweeness centrality. In social networks, for example, these are the people who have a lot of power to change messages that are being passed around. In transportation networks, such as an airline flight network, these nodes, or in this case airports, are the most vulnerable nodes for attacks and need to be well protected.

Until now we've only talked about centrality measures of vertices. But we could equally well talk about centrality measures of edges. For example, in many applications, such as road networks or power grids, it is important to determine the edges that could become bottlenecks. The importance of an edge can be defined equivalently to the betweenness centrality of a node by the number of geodesic paths that pass through this edge.

So to summarize, in this lecture, we have discussed two additional measures of centrality namely the closeness centrality and the betweenness centrality. Centrality measures can be used to identify important nodes or edges in a network. Now during the next lecture, we will move away from determining the importance of single vertices or edges, and discuss more general structural properties of the whole network.

5.2.4 Global Network Statistics

So, welcome back. In this lecture we discuss various structural characteristics of networks. And how these characteristics can be either similar or different in networks from different application domains. So we've already introduced the degree of a node. However, just knowing the maximal or minimal degree of any node in the network does not really say much about the whole network.

It is more illuminating to look at the whole degree distribution of a network instead. So for each degree, K, we compute the fraction of nodes in the network that have degree K. Let's discuss how such a degree distribution would look like for a social network. So the distribution will be very skewed.

Most individuals have a moderate number of friends, say 100 or 200. However, the tail of the distribution is very long. Celebrities, like Taylor Swift, have many thousands of friends and lead to very high variance in the distribution. Other individuals are somewhere in between. It turns out that such long-tailed degree distributions are found not only in social networks.

Many different kinds of networks exhibit a similar structure, where most nodes have a moderate degree, but there are a few nodes with very high degrees. Such high degree nodes are also called hubs. They appear, for example, also in the World Wide Web or in many biological networks, such as gene regulatory networks.

Most genes only regulate a few other genes, but a few genes are hubs and regulate all kinds of genes around them. So mathematically speaking, degree distributions are often right-skewed. They have long tails that extend to the right representing few high-degree nodes. Such distributions are often better represented on a logarithmic scale.

On the x-axis we plot the logarithm of the degrees, and on the y-axis we plot the logarithm of the fraction of nodes of a particular degree. For many networks, something surprising happens. Namely, this log law plot often turns out to be linear. This means that the fraction of nodes of degree K, is proportional to the degree K to some negative power.

In this case, we say that the degree distribution follows a power law. In fact, many networks have a power law degree distribution. For example, the World Wide Web, the Internet, citation networks, metabolic networks, or protein interaction networks all have a power law degree distribution. Also it turns out that the exponent of this distribution is usually somewhere between two and three.

So although networks from different application domains might vary a lot in their size, both the number of nodes and also the number of edges, the degree distribution is often very similar across different application domains. Everything we've done so far has concentrated mainly on the analysis of vertices and their properties.

Let's now turn our attention to the edges in a network. The maximal number of edges in a simple graph with n nodes and no multi-edges or self-loops, is just the number of pairs that can be formed from the N vertices. So there's N choose 2 which = N x N-1/2.

The density of a network is the fraction of edges that are actually present in the network. Networks in different application domains can have very different behavior in terms of their density. Let's first discuss the example of a social network. So take a small social network, let's say of a school with 1000 students.

And compare this to the social network, such as Facebook, with 1 billion users. How do you think the densities will compare? Well, while within the school you might have 20 to 100 friends, on Facebook, you might have more friends, perhaps several hundred. However, the number of friends is not going to scale with the number of people in the network.

You're not suddenly going to have 1 million times more friends just because there are 1 million times more people in the network. Such networks with relatively many nodes compared to the number of edges, are called sparse networks. So social networks are examples of sparse networks. And the same holds for the World Wide Web, the Internet, or citation networks.

So what are then examples of dense networks, where the number of edges scales up with the number of nodes? Well in fact, most networks that we come across are sparse. It is difficult to think of dense networks. One example though are food webs. Let's think of different sized ecosystems.

So very simplistically speaking, larger animals tend to eat smaller animals and don't differentiate too much between different species. So zebra and an antelope are both equally delicious from a lion's point of view. So if you take into account more animals in the ecosystem, the number of edges will scale proportionately.

This is an example of a dense network. Another network property that relates to edges in the network is clustering. Clustering has to do with transitivity of a relationship. So for example, since Guy knows Justin and I know Guy, it is not guaranteed, but more likely than otherwise that I know Justin as well.

So in social networks, we expect to see many triangles. The Internet and power grids have fewer triangles. Here, triangles represent redundancy. So let's think about how to quantify this. In a network with N nodes, the number of triangles is maximized by the completely connected graph.

This graph has N choose 3 triangles, so the density of triangles is just the # of triangles in the network divided by the total # of possible triangles, namely N choose 3.

But does the density of triangles really quantify clustering? Let's think of a completely connected subgraph and a bunch of single nodes or pairs. I would argue that this graph clusters very well, although the density of triangles is not that large. These people out here, they don't know anyone in this large cluster.

So we'd expect that there are no triangles between them. So rather than using the density of triangles, we should use the # of triangles divided by the # of connected triples. This really quantifies transitivity. It captures the following number, namely, how many people that Guy knows do I know as well?

So the clustering coefficient is an interesting way to quantify the large-scale structure of a network, and can be very different for different networks. So now let's end this lecture with a third global network measure, namely the diameter of a network. So what is the diameter of a network?

Well, it is based again on the assumption that information flow between any two nodes takes the shortest path. Diameter of a network is the maximum length of a shortest path between any two nodes in the network. So, it's the maximal amount of time that you have to wait for information to get from any one person to another person.

Or in mathematical terms, it is the maximal geodesic distance in a given network. We've all heard of the notion of small-world, or six degrees of separation. Namely the idea that everyone in the world is connected to everyone else by at most five acquaintances. This is equivalent to saying that the diameter of the global network of acquaintances is at most six.

The origin of the idea of six degrees of separation, is an experiment performed by Milgram in the 1960s. In this experiment, participants were asked to get a message to a particular person, by passing it from acquaintance to acquaintance. And surprisingly, while some of the messages never made it, the ones that made it arrived in an average of six steps.

In fact, we see the small-world effect in everyday life. It explains why news or gossip can spread super fast in a social network. Think for example about how quickly the news spread of one of your friends getting married or having had a baby. So this brings us to the end of this lecture.

We discussed different ways to quantify or measure large-scale network structures. Namely, we've discussed the degree distribution, the clustering coefficient, and the diameter of a network. These three measures give complementary insight into the large-scale structure of a network. In the next lecture, we'll discuss different network models. And also analyze how they behave with respect to these global network measures discussed in this lecture.

5.2.5 Network Models

Welcome back. So far, we've discussed various ways to measure or quantify the large scale structure of a network. But how did these networks come to exist? Many of them started very small and gradually grew to the size they are now. If you think about how the Facebook network grew over time, people gradually joined the network, and edges were added when people became friends.

This seems like a rather different mechanism than the growth of a road network, and it stands to reason that the large scale structures are different as a result. In this lecture, we'll study network models and how they lead to some of the large scale we examined in previous lectures.

By doing this, we'll get some insight into whether or not these models are plausible formation processes for an observed network. Having good network models also helps us when studying processes on networks such as the spread of a disease. Network models allow us to make statements for many networks at once without having to analyze each network separately.

Random graph models are network models that have some specified parameters, but otherwise, the edges in the network appear at random. A very simple random graph model for simple undirected graphs is a model where the number of nodes and the number of edges is fixed, and you just place the edges uniformly at random.

Related to the simple model but easier to handle mathematically is the well-known Erdos-Renyi model. Instead of fixing the number of edges, we only fix the probability of an edge. This means that the expected number of edges is held fixed, but the exact number of edges can vary around that number.

The Erods-Renyi model is one of the best-studied network models with very nice mathematical properties. One can pretty easily compute the average clustering coefficient, the degree distribution, and the diameter of an Erdos-Renyi graph. For example, what is the probability of seeing a node of degree 10 in an Erdos-Renyi graph with 100 nodes where the probability of an edge is 10%?

Well, there are 99 possible neighbors, and 99 choose ten possible ways of choosing exactly ten neighbors. The probability that there is an edge to exactly these ten neighbors is 0.1 to the power of 10 times 0.9 to the power 89 since no other edges are allowed to be there.

The resulting degree distribution looks like this. So it does not look at all like the power law degree distributions that occur in many real-world networks. Similarly, one can also easily compute the clustering coefficient and the diameter of an Erdos-Renyi graph. It turns out that Erdos-Renyi graphs have very small clustering coefficients and very large diameters.

This is the case, because locally, Erdos-Renyi graphs look like trees. So while Erdos-Renyi models are easy to handle from a mathematical point of view, they are not very realistic models for most applications. In particular, in most applications, the presence or absence of edges is correlated. Since so many real-world networks across different application domains show power law degree distributions, let's think about ways of creating network models that have such degree distributions.

So the easiest way to do this is to fix not just a degree distribution, but in fact, the whole degree sequence. In other words, instead of only fixing the number of nodes and edges of a network, let's fix the degrees of each node. So consider this friendship network with ten individuals.

The configuration model is a random graph model that allows to create networks with the same degree as this friendship network. So what is this model? Each node has a specified degree and is represented by a point with its degree mini stubs attached to it. Guy, for example, has many friends.

He has a high degree represented by the many stubs attached to him. Now, you match the stubs uniformly at random. Meaning, you choose a pair of stubs at random and draw an edge between them. Then you choose another pair of stubs from the remaining stubs and connect them by an edge, etc.

So this creates a graph with the desired degrees. However, this network model has one issue that might not make it realistic for some applications. Namely, the resulting networks can contain self loops and multi edges. For example, you see here, that Guy befriended himself, and I'm now double friends with Justin.

While self loops or multi edges are realistic for road or neural networks, this is problematic for modeling a friendship network, the power grid, or a telephone network. One option when using the configuration model for such applications is to just remove all self edges and multi edges. In fact, it can be shown that for large network sampled from a configuration model, the number of of such edges is vanishingly small.

So this can be a good approach in practice. Apart from the degree distribution that can be chosen as desired, let's analyze other large scale structural properties of networks sampled from a configuration model. One can pretty easily compute a clustering coefficient. It turns out that similarly to Erdos-Renyi graphs, the clustering coefficient is very small for large network sampled from a configuration model with fixed degree sequences.

So while the configuration model can achieve power law degree distributions, the resulting networks are locally tree-like and have small clustering coefficients. So how can we create random graph models with large clustering coefficient? One idea is to start off with the graph that has a high clustering coefficient, and then add in some randomness on top of that.

So circle and graphs are examples of graphs with high clustering coefficients. So how did these graphs look like? So you start off with all the nodes arranged around the cycle. And now, you connect each node to all its neighbors that are at some distance D. For example, take a ten cycle and connect each node to the two nodes that are two hops away along the cycle.

You see that this creates a graph with many triangles, so it has a high clustering coefficient. However, this looks far too structured to be a friendship network. In addition, the degree of each node is the same, so there is no power law degree distribution. Also, the diameter of this network is very large.

It does not show the typical small world connectivity. Is there a way to remedy this problem? Well, in order to reduce the diameter, we need to introduce some short cuts. One way this can be done is by choosing some edges at random and rewiring them at random. Now, doesn't this already look much more like a real friendship network?

What is really interesting is that rewiring even just a few edges reduces the diameter of the network greatly, and you can obtain networks that have a high clustering coefficient, and at the same time, also a small diameter. So such models give insight into why the small world property is so common in real-world networks.

In real-world networks, even if the underlying network is highly-structured, there will always be these links that happen just by chance. These links are the one's that create the small-world property. Such models are called small-world network models. They provide important insight into the creation of the small-world property. However, one can show that the degree distributions of networks in this model do not follow a power law.

So what kinds of processes can lead to a power law degree distribution? Why do we see this phenomenon so often in real world networks? We discussed the configuration model as a way of building a network that has a desired degree distribution. However, this model does not give us insight into the process that led to this particular degree distribution.

Let's look at the situation where power law degree distributions arise. In the 1960s, a researcher named Price studied citation networks. Here, the nodes are papers, and there is a directed edge from paper one to paper two if paper one cites paper two. Highly cited papers gain more attention and get read more often.

So it is plausible to assume that the probability of citing a paper depends on its existing number of citations. This logic leads to a network model called the preferential attachment model. The model is sequential, nodes are added one after another, and an edge is included between the new node and an old node with probability proportional to the degree of the old node.

For citation networks, this means that a new paper cites existing papers with probability proportional to their current citations. To make sure that any paper has a chance of being cited, each paper gets some probability for free. Interestingly, it turns out that preferential attachment models lead to networks with a power law degree distribution.

In addition, the exponent of this distribution is at least two, which mirrors the exponents observed in many networks, such as the world wide web, the internet, or metabolic networks. So an intuitive process, such as individuals preferring to befriend other individuals who already have many friends, can lead to power law degree distributions.

In this lecture, we've seen various network models. It became clear that every model has different strengths and weaknesses, and the choice of the model depends on the application. Network models are useful in order to find unexpected features in an observed network by comparing the observed network to a particular network model.

Most importantly, we saw how network models can provide us with insight into why the small-world property and the power law degree distributions are so widespread in real-world networks. Finally, network models also allow us to make statements about the whole class of networks at once. During the next lectures, Guy will discuss processes on networks.

5.2.6 Stochastic Models on Networks

Hi, in the last several lectures, Caroline has discussed various aspects of network structure. In this lecture, we're going to introduce a basic class of processes on networks called epidemic or contagion models. We can think of these processes as modeling propagation of information: disease, behavior or opinion. The spread of Tweets via re-Tweets would fit within the information category with Tweets propagating through the Twitter graph.

Or you might think about a funny joke told from one friend to another, which propagates through society by a social interaction on a friendship network. In a similar way, a virus, such as the flu, propagates from one person to another. In this case, the underlying network might connect to people, if they spent time in close proximity, or in the same room.

Computer viruses are from the mathematical perspective, not all that different from biological viruses, and spread from computer to computer through the Internet. While we don't necessarily think of behavior as contagious. Humans are social creatures and various behaviors do propagate it through our social network. Our eating or exercise habits, political views, product or fashion choices, all propagate through the network of social interactions.

This raises some basic questions. What sorts of models make sense for such processes? Can we make predictions about how a process will spread? What does this depend on? How many nodes will be infected? If we observe a process that has just begun, what is the probability that we get an epidemic that spreads to a global scale?

The answers to these questions depend on characteristics of the process and on properties of the underlying network. While the terminology comes from epidemiology, or the study of disease, an epidemic isn't necessarily a bad thing if the process is modeling something like music choice or good exercise habits. An epidemic can also describe the spread of opinions such as whether or not global warming is a serious issue.

The specific details of a contagion process are often very complicated. For instance, the biology of your immune response plays a role in determining whether you will fall ill with the flu. Modeling the dynamics of a spreading virus by incorporating every possible detail is hopelessly complicated. So instead, people study simplified models that capture the main properties.

The models we'll be interested in capture the macroscopic, network-level behavior. Perhaps the simplest model is what is known as the SI model. S stands for susceptible, and I stands for infected. At any point in time each node is susceptible or infected. In the simplest setting, nodes deterministically infect their neighbors.

A better model is probabilistic, where each infected node infects each of its susceptible neighbors with some probability P. We can think of the spread of information through an online social network. The value of P is high for something really interesting. Such as a truly hilarious parody of a presidential candidate.

Conversely, the value of P is lower for a boring an uninspiring parody. One way to think about this model is that each of my friends who gets infected have an independent opportunity to infect me. As long as I'm not yet infected. There is just one shot for the infection to spread across each edge.

Let's suppose that a random node in the network is infected. You might want to predict whether a given trend will take off. Or whether a YouTube video will go viral. Obviously this depends on the value of P which captures how contagious the trend is. But it also depends on the network structure.

If everybody has many friends, we'd expect things to spread more widely. This is especially clear in the context of the flu. The network describing contact between kids in schools is very dense. So if one person gets sick, it's not unlikely that everyone gets sick. For sparse networks where most nodes have relatively low degree, we can get some intuition by thinking about a value $P = 1/3$.

This means that each node adjacent to an infected node has probability one-third of being infected by the infected node. Let's go back to the first infected node, and think about the progression of the SI model happening in rounds. If this initial node has six neighbors, then on average two of them will be infected.

If each of these have six additional neighbors, then on average, each will infect two more, for a total of four at the second round. Since each node infects two nodes on average, the number of infected nodes grows by a factor of two with each round, and the four nodes will, on average, infect another eight.

Pretty quickly, a huge number of nodes in the network has been infected. One thing worth mentioning about the calculation we just did is that after the first node, we assumed each node has six additional neighbors beyond the one that infected it. So if we count this infecting node, we see that the average degree is actually seven.

Suppose the infection probability P is still one-third. But the average degree is low, with nodes only having three neighbors on average. Then the expected number of neighbors a node can infect once it becomes infected is two neighbors times one-third probability each which is equal to two-thirds. Each newly infected node on average infects fewer than one new node, so the process inevitably dies out.

This intuition of comparing average degree to the probability of infection, holds in a fairly general setting, as long as each of the neighbors infects me independently. It holds in network models including the configuration model, which Caroline described. Roughly we can multiply the average degree in the network and the probability of infection across an edge to get the branching number of the infection.

If it's greater than one the process has the possibility of spreading to a macroscopic fraction of the population. And if it's less than one, the process dies out. In the SI model, once a node is infected, it remains infected forever. One natural way to extend the SI model is to allow the possibility of nodes to recover.

This makes sense if we think about recovering from the flu. Once you're recovered from the flu, your body has developed immunity to that strain, and you won't catch it again. So in the SIR model we have an additional state which we call recovered. Yet another model is called SIRS.

Here an infected node recovers and has immunity but only temporarily and then goes back from the recovered state to being susceptible. This is a reasonable model for the spread of fashions. Where the same fashion comes back again and again and again. I like to rock climb, and believe it or not, in the 80's it was very fashionable to wear hot pink Lycra while climbing.

A decade later, nobody could believe how crazy people looked. But now I'm starting to see more and more climbers wearing bright colored Lycra, and I might have to get some Lycra pants myself. Although, I may chose to go with neon green instead. In the next lecture, we'll discuss how we can use the network structure for interventions.

Which nodes we advertise to in order to maximize the spread of the process. While we obviously don't wanna spread a virus if we're thinking about the spread of healthy habits, such as regular exercise. It is definitely useful to understand how to maximize the spread of such behaviors. And of course, any company would like to know how to maximize the effect of word of mouth advertising.

5.2.7 Influence Maximization

Welcome back. In the last lecture, we talked about modeling the spread of a virus, a piece of information, or an idea. We discussed the SI model of contagion, which stands for susceptible or infected. Each infected node has one shot to infect each of its susceptible neighbors, and this occurs independently for each neighbor with some probability P.

In this way, the process spreads through the network starting with some initial set of infected nodes A. For each set of nodes A, the infection spreads in a random way through the network, and eventually stabilizes with some final set of infected nodes. A very natural question arises. If we can choose some subset of individuals to target, how do we choose them to maximize the spread of ideas?

In other words, to whom should we advertise in order to maximize the size of the final infected set? Because friends recommending a product to their friends can have such a significant effect, viral marketing is really effective. This question, to whom should we advertise, is not just about making money since the same concepts are relevant to public opinion, spread of innovation or behavior.

For example, a climate change organization may want to change public opinion about whether or not global warming is a serious and urgent issue. It's interesting to think about how in the United States, smoking is viewed very negatively, despite being hugely popular several decades ago. In Europe things are very different, with many people smoking.

Whether you're lobbying in Congress to sway the opinion of political representatives, or running an ad campaign to change public opinion, understanding network effects is important. Let's modify our terminology and use inactive and active instead of susceptible and infected, since we're thinking about trying to maximize spread and I don't really want to maximize the spread of a disease.

We can formalize the influence maximization question like this. Suppose we can choose a set of K initial people to adopt the behavior or product, thereby making them active. Which subset should we choose in order to have the largest expected number of active nodes when the process eventually stabilizes?

Concretely, suppose we're giving away free tickets to the upcoming Justin Bieber concert to a few people in order to maximize the number of people who eventually buy tickets. A node is active if the individual has a ticket to the concert. Because the eventual set of active nodes is random, due to randomness in the contagion process, we look at the expected number of nodes that are active when the process stabilizes.

In this lecture we'll think of the influence of a subset of nodes as the expected number of nodes that eventually becomes active. Choosing a subset of nodes in order to maximize influence is complicated. On one hand, it makes sense to choose nodes that are fairly spread out so that propagation will occur in more parts of the network.

On the other hand, unless there's sufficient density of active nodes to get the process going, it might just die out. It's useful to think about choosing a single active node to maximize the spread. Luckily, there's a simple way to figure out how many nodes are expected to eventually become active if we have a good model for the contagion process.

We can simply perform an experiment by simulating the process spreading through the network. If we do a number of such experiments, and average the number of individuals that eventually become active, we get an estimate of the node's influence. This is known as a Monte Carlo approach. It turns out that we only need relatively few runs of this experiment, specifically a number logarithmic in the size of the network, in order to get a good estimate of a node's influence.

Doing this for each individual node in the network is relatively inexpensive. Suppose that there are N nodes and M edges. Simulating an instance of the contagion process requires that most M computations since each edge plays a role in the process it most wants. We do this Log (N) times for each of the N nodes.

So in total, we get on the order of M times N times Log (N) computational cost. Unfortunately, this approach doesn't work directly in order to choose K nodes. While we can simulate the process starting from any subset, just like for a single node, there are way too many subsets.

Roughly, there are N to the power of K subsets of nodes of size K. So if K is 50, and our network is the size 1,000, we're talking about 1,000 to the power of 50 sets that we would need to check. We'll never be able to search over all possible subsets to choose the best one.

A way to avoid searching over all subsets of size K is to incrementally build up our influential subset A in a greedy manner. The algorithm is called greedy because at each step it tries to make the biggest possible gain in influence. We start by adding to A the single most influential node, which as we've just discussed is computationally feasible.

Next, we find the most influential node given that nodes in A are active, and add it to A. The key here is that we can find this most influential node in exactly the same way as before by simulating the propagation. Not only this, but the search over individual nodes to add is efficient, because we just have to check in most end nodes.

The algorithm stops after K iterations when we've added K nodes to our set A. The total amount of computation is just K times the amount of computation for a single node, so on the order of K times M times N times log N. How good is the set A that we've ended up with?

It turns out that for quite general models of contagion, the influence of the set A is guaranteed to be almost two thirds of the best possible subset. Of course it's computationally unfeasible to find the best one, so this sort of guarantee gives pretty good confidence. And in practice, such greedy algorithms often work well.

Let's say a few words about more general contagion models. You might imagine a situation where the likelihood of a node being infected by a neighbor can change depending on which subset of neighbors has already attempted and failed to infect the node. Think of it this way. The latest Apple iPhone comes out and whenever each of my friends buys it, they tell me about how amazing it is.

Each subsequent friend telling me about their new phone does not result in me independently deciding to buy it myself. The tenth friend who tells me about the product will surely have less of an effect than the second friend. Nevertheless, more general contagion models, such as the so called Order Independent Cascade Model, can capture such effects.

Other models incorporate thresholds for an individual to model a situation where I may be more likely to adopt a behavior only after several of my friends have adopted it. In many real life situations, several processes are competing with one another. For example, different viruses are competing simultaneously in your body.

And videos are competing for our attention as they propagate through Facebook. This makes things significantly more complicated, and it is still very much an active area of research.

5.3 Graphic Models

5.3.1 Introduction to Undirected Graphical Models

Welcome back. We'll now have three lectures on graphical models. This lecture is on undirected networks and the processes on them.

We will introduce a natural dynamical process on an undirected network and observe how this gives rise to joint distribution over the nodes of the network. Such a joint distribution is known as an undirected graphical model.

In the next lecture, we'll study the two most important undirected graphical models. Namely, the Ising model, and the Gaussian graphical model. Finally, in the third lecture on graphical models, we'll talk about how to learn the underlying network from process data.

We will see later that the same graphical models framework can be used to make predictions from partial information.

Undirected graphical models sound very useful. How should I start thinking about these models?

Let's start with undirected network, for example, consider a social network such as Facebook. Remember that the graph is undirected, because friendship is bidirectional. Let's consider the gaming industry, we all know it's a multi-billion dollar industry. So it would be great if we were able to predict how a game spreads in a social network or who is likely to play.

So let's consider a game that costs $1 to play each day. This game is played online, so it's possible to compete directly with your friends. It's more fun to play if your friends are playing too. But at the same time, if relatively few of your friends are using it, then it may not be worth spending $1 that day.

Yes, and some people may really like the game and will play no matter what. And other people are busy and may give it a try only if many of their friends are playing.

Okay, so each day each person decides to play or not, and there is some randomness. Although having friends that play the game is an incentive to play, it's not a deterministic process. And we'll imagine that this process continues for a while.

Now we'll observe a snapshot of the state of the system on some particular day. So who is playing that day, and who is not playing. This corresponds to a list or a vector with one binary entry per person, 1 or 0, corresponding to play or didn't play that day.

Perfect, so we have this vector. Whether I played or not depends on whether or not my friends played and some randomness. If I didn't have any friends at all, what I do is independent of everything else in the network.

But luckily you have many friends.

Okay, so what each person does depends on what their friends do and this process evolves over time.

At the beginning, when a new game is launched people try it out randomly. The network structure does not play a role yet. However, if you let the process evolve long enough, then the distribution over the social network and who is playing, and who isn't playing, reaches a steady state.

A graphical model precisely captures the joint behavior, or joint distribution, of the process once it has reached a steady state. So to be more specific, imagine you get a binary vector of who played and who didn't play that day for many different games. So you have one binary vector for each game.

Then the interaction between individuals in all of these vectors is governed by the same underlying social network.

I see, so each particular vector from different games can be very different. But the statistical dependence between the people playing or not is the same for different games.

Exactly, the joint distribution inherits its statistical structure from the underlying network. Now, suppose I don't have information about a particular user, you, for instance, and I want to predict whether you will play today.

Well, whether or not I play will be based on whether my friends play, so you could use this information to make a prediction.

Yes, and once I know whether each of your friends played or not. Additional information about other people in the network does not help me make my prediction. This is because the effect from someone multiple hops away in the network must pass through your friends in order to have an effect on you.

Put differently, a given node's value is statistically independent from the rest of the nodes conditional on its neighbors.

This is known as the Markov property or, more specifically, the undirected Markov property. It turns out that under mild conditions, this implies a more general statement, namely that any two sets of nodes are conditionally independent, given the values of a set of nodes that separates them.

To be more precise, a set of node C separates A and B if all paths from A to B pass through C.

It's a quite remarkable fact that under mild conditions, such as the positivity of the distribution, the local statement that a node is conditionally independent of the rest, given its neighbors, is equivalent to this global statement in terms of graph separation.

So we've discussed how conditional independent statements correspond to separation statements in the graph.

However, it is important to emphasize, the two nodes can be highly dependent, yet still not have an edge between them because they interact via other nodes. Edges in a graphical model capture only direct interaction.

I see, so let's be more specific. You and Justin Bieber are not, maybe yet, friends in real life, so you are not connected by an edge. However, whether you and Justin play a particular game could be dependent, since you might have common friends. However, since you don't know each other personally, we can always find a set of people that separate you from Justin. And conditional on these people, whether you and Justin play a certain game, becomes independent.

To summarize, a graphical model describes the joint distribution of random variables with the nodes of the graph. And variables satisfy the Markov property with respect to the graph.

One way such distributions arise is through processes on networks. In our example, the joint distribution of people playing a particular game is described by a graphical model.

Because the interactions occur through the links of the network, the graph underlying the graphical model that describes the conditional independence of the joint distribution is the social network itself.

In the next lecture we'll see two specific examples of graphical models, and get some insight into the power and flexibility of these models.

5.3.2 learning Undirected Graphical Models From Data

Welcome back. During the last lecture, we've introduced the concept of a graphical model, and we've discussed the example of gaming in a social network.

In a graphical model, the nodes of the network correspond to random variables. And the edges represent dependencies between these random variables.

In the gaming example, the data are binary vectors, one vector per game, corresponding to a snapshot of game subscription over the whole social network.

Caroline, for example, loves to play Halo, so she would have a one in the vector corresponding to the game Halo. Probably many of her friends play Halo, as well.

I know, you're more of a slow-paced Settlers kind of person. It looks like we'll never be able to be friends.

Aw. These binary vectors are samples from a joint distribution over all the nodes in the social network. Joint distribution is special because it inherits properties from the underlying social network. Nodes that are connected by paths in the network are dependent since each person's decision of playing a certain game is influenced by their friends.

Hm, it's so much more fun to play Halo cooperatively with my friends. The new version can be played with up to three other players. I have much more incentive to play when at least two other friends play as well. Can we take this into account in a graphical model?

Indeed, the graphical model is very versatile, and one can easily include higher order interactions. In fact, modeling with higher order interactions can be important. Here's an example. I have two friends who always argue. I love going to parties with each one of them separately, but I stay away from a party if I know that they will both be there.

But in your example, the three-way interaction has a very different effect than for me when playing Halo.

How so?

Well, interactions have weights. I am more likely to play Halo when I know that two of my friends are playing.

While it is important to take into account higher order interactions in some applications, very often it suffices to consider simple models that only have pairwise interactions.

This means that the interactions are only on the edges of the underlying graph. For binary data, such models are called Ising models. Ising models were introduced to model ferromagnetism in statistical mechanics, and are now used all over the place. They are used to model neural spike patterns, likes and dislike of YouTube videos or Facebook posts, economic modeling, signal processing, computer vision, image segmentation and many other areas.

Well, Ising models are really good models for binary data. You wouldn't want to use Ising models for continuous data. For example, for weather forecasting, I'm pretty sure nobody would be happy with a binary forecast.

The only possibilities would be sunny or downpour.

Exactly, for modeling continuous data, Gaussian graphical models are used everywhere. For example, for wind speed forecasting, modeling the interactions in the stock market, or in biology for modeling the interactions between genes and their expression.

It makes sense that Ising models are used in a binary setting. But why is the use of Gaussian graphical models so widespread for modeling continuous phenomena?

Well, the Gaussian distribution with it's nice bell-shaped form is easy to handle mathematically. In addition, the central limit theorem from probability theory tells us that averages of random variables are normally distributed when the number of random variables is sufficiently large. This implies that physical quantities that are expected to be the sum of many independent contributions often have distributions that are nearly normal.

So this is why people's height is approximately normally distributed. Height is believed to be the sum of many independent contributions from various genetic and environmental factors.

Yes, and similarly for measurement errors, gene expression and wind speed.

Let's discuss in more detail how we could model of wind speed using a Gaussian graphical model. We have data from different weather stations across the country, so the weather stations are the nodes in the graph. What are the edges?

Well, assuming that we know the wind speed in New York, then the wind speed in Washington wouldn't provide me with any further information about the wind speed in Boston given what I already know from New York. So wind speed in Boston and Washington are conditionally independent given wind speed in New York.

That makes sense. So it seems reasonable to assume that a node is only connected by an edge to the closest nodes geographically. By the Markov property this implies that the weather in two cities is conditionally independent given the weather in all cities between them. This seems reasonable.

Well, but maybe we have to be a bit more careful. What if two cities are spatially close by, but they're separated by a big mountain?

Then the weather in these two cities is probably conditionally independent, given the weather in all the cities around the mountain. So we would remove the edge connecting the two cities, although they are spatially close by.

So for wind speed forecasting, we can intuitively build a graphical model that represents the conditional independence relations that we expect to hold. While a meteorologist might be pretty good at doing this, we would certainly fail miserably.

Using our graphical model would probably be a disaster. So in the next lecture, we will present principled ways for learning the underlying graph from data that does not require domain expertise.

We end this lecture by pointing out a deep property of Ising models and Gaussian graphical models.

It is known as the maximum entropy principle. This result tells us that the Ising model and the Gaussian model are extremely flexible models. In fact, they can capture any pairwise correlation structure that can be constructed for binary or for continuous data. Not only that, but they are the least constrained, in a precise sense.

So although the Ising model and the Gaussian graphical model are very simple models, this result explains why they're able to model very complicated phenomena, and therefore are used in so many application domains.

5.3.3 Ising Model and Gaussian Graphical Model

Hi, in this lecture we're going to talk about methods for learning Ising models and Gaussian graphical models from data.

Suppose you have access to data consisting of ratings, like or dislike, for 50 YouTube videos, from everyone at the university.

Okay, and let's assume for simplicity that everyone has rated each of these videos. So we have binary vectors, where each vector contains the ratings of each person for a particular video. We'll assume that everyone has rated all 50 movies.

Because the data is binary and people are influenced by the people they interact with, it makes sense to model this as arising from an Ising model

Wait. What graph should we use to describe this model?

That's exactly the question. We need to learn the graph describing who interacts with whom.

I see. And we then also need to learn weights for the edges corresponding to strength of interactions.

That's right. But it turns out that it is often possible to first learn the graph structure and then it is relatively easy to determine the appropriate edge weights.

Perfect. So we have all of this data with each person corresponding to a node in a graph and we need to figure out where to put the edges. Let's add edges between people whose ratings are very similar.

Well, this is where things get subtle. Just because two people's ratings are very correlated, this does not necessarily mean that they directly interact.

It could be the case that they simply have many friends in common. More generally, effects can propagate through the network, so people who are far apart can still be highly correlated.

So what can we do? It's seems quite difficult to find the actual structure of interactions?

Well, we know that the graphical model satisfies the local Markov Property. What this means is that your likes and dislikes are conditionally independent from all the other people's ratings given your friends ratings.

So if we want to test whether a particular set of people includes all of your friends, we can try to test whether the rest of the nodes are conditionally independent of you given this set.

Computationally, this procedure is only feasible if the size of each neighborhood is relatively small, meaning that every person does not have too many friends, since this involves searching over quite a few subsets.

Hm, but this assumption seems problematic. Some people have a lot of friends, like Justin Bieber.

Yeah, that's right. So, let's think of other ways how we can determine that there is no edge between two nodes.

By the Global Markov Property, two nodes are conditionally independent given a set of nodes that separates them. In particular, if there is no edge between two nodes they must be conditionally independent given all other nodes in the network.

So this gives rise to an algorithm. We start with the complete graph, where everybody is connected to everybody else. Then we test, for each pair of nodes, whether they are conditionally independent, given all the other nodes. If so, we remove the edge. Otherwise, we keep it.

This can be done easily for Gaussian graphical models.

Specifically, one can estimate the covariance matrix from the data, then invert it. Any entry I, J in the inverse covariance matrix is 0 up to noise, precisely when I and J are conditionally independent given all the other nodes in the network. So we can use the inverse covariance matrix to estimate the underlying graph.

The entries that are close to zero can be assumed to be just noise, and correspond to the lack of an edge.

Okay, this nicely shows that there is a trade-off between the number of samples and how large an edge weight has to be so that such an algorithm can detect it as an edge. If we have a lot of samples, we can differentiate between the missing edge at the value zero and the present edge with a very small weight like .05.

Let's make this precise. Let's take a network with P nodes. We've already discussed that it is more difficult to estimate graphs that contain people like Justin Bieber.

So let's denote the highest degree of any node in the network by D. It is known that if a number of samples is at least D squared times the logarithm of the number of nodes P, then one can detect edges that have weights as small as the square root of log (P) over N.

That's a very nice result that holds for Gaussian graphical models.

Why can't we do something similar for Ising models?

The difficulty with Ising models is due to having binary values. If many of your friends influence you rather strongly, then whether you take value zero or one will be more or less determined by them.

In that case, you'll be influenced very little by one additional friend with a weak edge. You can think of this as a saturation that can occur in binary models. In Gaussian models, a weak neighbor will have a more noticeable effect since a continuous value always has the freedom to increase.

Okay so let's return to your original idea of trying to learn the friends of each person.

Turns out that there are methods that can do this and work well for both Ising models and Gaussian graphical models. These methods are called greedy methods because they try to make as much progress as possible with each step.

The basic idea is to try to determine the neighbors of each node without searching over old subsets. Let's imagine we're trying to determine who are your friends.

So let's keep track of a set S that we think of as possibly being my friends. We'll simply start by adding the node that contains the most information about me.

We next add the node that contains the most additional information about you, given what we've already learned from the one node in S.

I see, so we can keep going in this way adding at each step adding the node having the most new information about me beyond what's already there in S. But when do we stop?

Well, once our set S contains all of your friends, then each other node is conditionally independent of you given S. That's because of the Markov property. The reason why this works well is that each node added to S contains useful information about you, so at every step a certain amount of progress is made.

Okay, so S may contain a bunch of nodes that are not my friends but not too many. And by the Markov property, each of the extra nodes in S that are not actually my friends are conditionally independent of me given S, so they can be removed in the final step.

Yes, great. You can now do this for each node, to figure out its neighborhood. This specifies all the edges that we want to add to the graph.

So to find my friends, we added a bunch of nodes in a greedy fashion according to information gained, and then at the end we removed the nodes that were added incorrectly.

There are other ways to do this that often work well in practice where you alternate removing and adding nodes. The key to these greedy methods is that progress is made at each step. Because these methods add one node at a time, they're very fast and you can scale them to very large graphs.

5.3.4 Introduction to Directed Graphical Models

In the last few lectures, we discussed undirected, graphical models and how to learn the underlying network structure in these models. We discussed friendship networks, and observed that friendship is usually by directed. If I call you a friend, then you usually call me a friend as well. So it makes sense to model social networks as undirected networks.

However, in other applications, we might care about the direction of the relationship. In fact, in many applications we are interested in determining cause, effect relationships. During this and the next lecture, we will discuss directed graphical models. We talked about weather last time, so let's start by discussing a small example related to weather.

Supposed that we measure five variables in the course of a month. Namely maximum daily temperature, average daily wind speed, amount of daily solar radiation, average daily cloud cover and average daily ozone values in Boston. They are directed relationships among these variables, for example, ozone pollution has created near the Earth's surface by the action of daylight UV rays on ozone precursors.

In addition, during heat waves ozone values rise because plants can't absorb as much ozone. Finally, high wind speed disperses ozone precursors, and hence, often leads to lower ozone values. Assuming wind doesn't transport ozone into town. So we obtain a directed graph with radiation, temperature, and wind pointing to ozone.

Should there also be a directed edge pointing from cloud cover to ozone? Well although ozone values are associated with cloud cover this association is most likely form through radiation. So we would not add the directed add ship. This directed graphical model represents our belief that ozone values are a function of radiation, wind speed and temperature plus some noise.

And that knowledge about cloud cover does not increase our ability to predict ozone values. If we already have access to data on radiation, temperature, and wind speed. So in this example, we made use of the main knowledge to construct the graph, but there might also be other edges connecting these variables that we missed.

Since I don't have the necessary domain expertise to be sure the model is a good one I would not want to predict the ozone levels based on this limited model. During the next lecture we will discuss methods for learning the underlying directed graphical model without requiring any domain expertise.

So let's discuss another example where the direction of the edges in the network is crucial. There is a lot of talk about gene therapy these days. In order to develop a gene therapy successfully, one needs to have a very good understanding of how a gene affects all the other genes.

One can measure the expression of each gene in many individuals, meaning how much protein each gene produces. In order to develop successful gene therapies, one needs to know which genes up or down regulate each other. This can be represented by a directed network where the nodes are the genes and a directed edge pointing from gene one to gene two means that gene one up or down regulates gene two.

For developing gene therapies just knowing the underlying undirected gene association network is definitely not sufficient. So in these two examples we have hinted at what a directed graphical model is. Just ask for undirected graphical models each node in the network is associated with a random variable. The directed edges describe dependencies between the random variables.

Or to be more precise, like for undirected graphical models, missing edges represent conditional independence relations. So what kind of joint distribution on the random variables does a directed graphical model encode? It's actually a very intuitive model, the joint distribution over all the nodes and the network, factors in such a way that each node only depends on its parents.

Meaning, that the distribution of each random variable is only a function of the nodes that point into it plus some noise. So how can we use this information to learn the underlying network? This definition says that if I could go into the network and intervene at a particular variable then this will only have an effect on old variables downstream.

So by performing such interventional experiments, we could find out which nodes are downstream from another node. And little by little, learn the underlying directed graph. Performing such interventional experiments is possible when studying, for example, gene regulatory networks. One can knock out a particular gene, meaning one makes this particular gene inoperative and one can then analyze how the expression of all other genes changes.

If the expression level of a gene B changes when knocking out gene A, this means that gene B has to be downstream from gene A. Gene B might not be connected to gene A by a directed edge, but there must be a path from gene A all the way to gene B.

Possibly through gene C, D, E and F. And the expression of all these genes on the path changes as well when we intervene on gene A. Such gene knock out experiments are routinely performed in order to determine some of the links in the gene regulatory network. However, a human has about 20,000 genes and one can imagine that knocking out one gene after another is pretty laborious.

So for large networks, it is difficult to perform interventional experiments to determine the complete underlying directed network. In other applications, it is not even possible to perform interventional experiments. Think about the weather application. If we could keep radiation, temperature, and wind speed fixed and vary the cloud cover, we could very easily determine if there is an edge pointing from cloud cover to ozone.

However, tuning variables in such applications is not possible, and we have to rely on observational data in order to infer the directed edges. Similarly, in various applications related to human diseases, although theoretically possible, it is ethically inappropriate to perform interventional experiments. For example in the documentary Super Size Me, Morgan Spurlock performed an experiment on himself.

He ate three times per day at McDonald's for a whole month and analyzed the effect of fast food on his physical and psychological well-being. Well Morgan Spurlock decided for himself to perform such an experiment. It would be ethically problematic to perform a similar interventional experiment and assign a group of people to eat at McDonald's every day.

This shows the need for methods that can learn the underlying directive graphical model from observational data. So we've discussed various methods for learning an undirected graphical model using the undirected Markov property. Remember that the Markov property describes the conditional independent's relations induced by the graph. Now is there a similar notion for directed graphical models?

So for directed graphical models the so called directed Markov property allows us to read off the conditional independence relations that hold in the joint distribution directly from the graph. The directed Markov property basically says that any node is conditionally independent from its ancestors, given its parents. So to illustrate this, let's return to the weather application.

The missing edge between cloud cover and ozone means that ozone values and cloud cover are independent given the parents of ozone, namely radiation, temperature and wind. While this is a local statement similarly as for undirected graphical models there is also an equivalent global statement which describes when two sets of nodes are independent given a third set of nodes based on separation statements in the graph.

To summarize, in this lecture we introduced directed graphical models, discussed examples of applications of directed graphical models and introduce the directed Markov property. During the next lecture, we will introduce also the global version of the directed Markov property, and discuss methods for learning the underlying directed graph from observational data.

5.3.5 Learning Directed Graphical Models From Observational Data

Welcome back. In the last lecture, we introduced directed graphical models and the directed market property. In this lecture, we will discuss how to learn the underlying directed graphical model from observational data, and what some of the problems are that we encounter when doing so. Let's start off with another example of a directed graphical model, where the underlying graph structure is clear from the beginning.

Here is the Bush family tree. Each node corresponds to a person, and a directed edge from node I to another node J means that node J is a son or daughter of node I. To each node or person, we attribute a random variable. Let's assume that each random variable corresponds to genetic information about that person.

The local directed mark of property says that the genome of each person is independent of the genome of all its ancestors, given the genome of its parents. This makes sense. For example, the genome of former President George W Bush is independent of the genome of his grandfather, given his parent's genome.

In fact, even conditioning only on his father is sufficient to make Bush Jr. independent of his grandfather. Because knowing his grandfather's genome does not provide any further information about Bush Jr.'s genome, other than what we already know from his father's genome. This shows that the directed graph encodes more conditional independence relations than the ones given by the local directed Markov property.

Let's try to establish further conditional independence relations from this example. Note that without conditioning, grandfather and son would be dependent, since they are related. Similarly, the genomes of former President George W Bush and his brother, Jeb Bush, are dependent, since they are related. However, once we condition on the genomes of their parents, then they become independent.

Because knowing Jeb's genome does not give you any further information about George W's genome, other than what you already know from their parents. We see that just as for undirected graphical models, conditioning on the nodes that lie on the path between two nodes renders the associated random variables independent.

However, unfortunately, for directed graphical models, this is not always the case. Consider Bush Sr. and his wife, Barbara Pierce. Most probably, their genomes are independent, since they are unrelated, so these two nodes are independents, although there is a path from Bush Sr. to Barbara Pierce through their children.

However, once we condition on one of the descendants, for example, on Bush Jr., then the parents' genome becomes dependent. Because everything that Bush Jr. has, but his mother doesn't have, comes his father. Related to this observation is also the concept of explaining away. So suppose we observe that Bush Jr. has a certain genetic trait which is relatively rare.

Say a specific genetic variant that allows him to jump very high. We know he got it from one of his parents, so the probability of Bush Sr. having the gene is much higher if we know that Jr. has it. And likewise, the probability that Barbara has it is higher knowing that Jr. has it.

But if we find out that Barbara has the jumping gene, it explains why Jr. has the gene, and it's now unnecessary for Bush Sr. to have it. Conversely, if Barbara does not have it, we know that for Bush Sr. definitely had it, since Jr. had it, and he needed to get it from somewhere.

In this way, there is dependence between Bush Sr. and Barbara, conditioned on Bush Jr. having the jumping gene. Consider another example where the nodes are just binary, namely whether it rained last night, whether the sprinkler was on during the night, and whether the lawn is wet in the morning.

It is obvious that in this example the underlying directed graphical model looks like this. While whether it rained last night, and whether the sprinkler was on, can be independent events, when conditioned on the fact that the grass is wet, they become dependent. Knowing also that it rained last night, the probability that the sprinkler was on as well is reduced.

This common and intuitively compelling pattern of reasoning is called explaining away. So we've now seen two examples of directed graphical models where conditioning can make nodes that were previously independent, dependent. In fact, this happens whenever you condition on a descendant of the two nodes. This structure, with two nodes having edges directed to a third node, in other words, having a common child is called a v-structure.

So all these examples of conditional independence and dependence illustrate how a directed graphical model, similarly to an undirected graphical model, encodes a global set of conditional independence relations. Basically, the global directed Markov property asserts that two nodes are conditionally independent if on every path that connects these two nodes, there is either a descendant that is not conditioned on, or a non-descendant that is conditioned on, exactly as we have seen in the examples.

So this tells us how to read off conditional independence relations from a directed network. However, what if we don't know the underlying network and we would like to learn it? So suppose we're given gene expression data, and we would like to learn the underlying gene regulatory network. Now, from gene expression data, we can learn the conditional independence relations.

So then the question becomes, how do we go from conditional independence relations to the directed graph? So similarly, as for undirected graphical models, we can use the conditional independence relations to learn the underlying graph. So we start with a complete undirected graph where everyone is connected to everyone.

Now, if nodes I and J are conditionally independent, given some subset of the remaining nodes, then we remove the edge between I and J. This procedure results in an undirected graph. Now, some of the edges can be directed by finding pairs of nodes that are independent, but become dependent by conditioning on a node between them.

Since conditioning can only make nodes dependent if the condition variable is a descendant. So what are the problems with this algorithm? Well, first of all, it will not be able to orient all the edges. While it can infer the v-structures, if we have only two nodes, say smoking and cancer, it cannot decide whether the edge points in this direction or in this direction.

In this two-node example, the only thing that the algorithm is able to infer is that there is an edge between smoking and cancer. In fact, this is not a shortcoming of the algorithm, but has to do with the nature of the data. From observational data on two nodes, it is impossible to decide the direction of the edge.

One would need to perform an interventional experiment instead. Second, to use this algorithm, we assume that there are no unobserved nodes. For example, nowadays, it is widely accepted that smoking causes lung cancer. However, only a few decades ago, there was a large dispute about this. Opponents argue that there is a hidden, or yet, unknown genetic factor that makes a person more prone to smoking, and also more likely to develop lung cancer.

A hidden cause could explain the apparent correlation. The occurrence of hidden, or unobserved variables, makes it very difficult for any algorithm to learn the underlying network. Edges in the network could always be there. Just because of a hidden variable, that affects both variables and hence, introduces correlation. If the goal is to learn the underlying network, it is therefore important to take into account as many variables as possible, and then be prudent about interpreting the resulting network.

Finally, there is another problem besides appearance of additional edges in the network due to hidden common causes. The algorithm could miss out on edges because of cancellation of causal effects. As an example, consider a particular medicine that cures a certain lung disease, so it has a positive effect on lung functioning.

However, at the same time, this medicine is extremely strong, and it has a negative effect on the immune system, which itself has a negative effect on lung functioning. It is possible that for certain medications, these positive and negative effects can cancel each other out, leading to data sets where lung functioning and taking the medication are independent.

This would correspond to a directed graphical model where medicine and lung functioning point to the immune system. So not only did we miss out on an edge, we also made a mistake with respect to the orientation of the edges. So to summarize, in this lecture, we discussed ways of learning the underlying directed network from data on the nodes.

In particular, we showed that this problem is an inherently difficult problem, especially when we only have access to observational data. Interventional data provides more insight into the structure of the underlying network, but might be impossible or difficult to obtain. If we learn the underlying directed network purely from observational data, it is important that we're prudent about interpretation of the network, specifically with respect to hidden common causes and cancellation of causal effects.

5.3.6 Inference in Graphical Models - Part 1

Hi, In the last several lectures we've introduced graphical models. We've seen that they're flexible in powerful way to model complex systems including networks. We also talked about learning graphical models from data. We'll suppose that we have either a learning model from data or alternatively we have domain knowledge that has allowed us to specify a model.

In either case, in the next couple of lectures, we will assume that we have a known graphical model. Graphical models are useful, not only because they are flexible in modeling network phenomena, but also because they facilitate answering statistical questions. Specifically, there are a variety of fast algorithms that have been developed over the last few decades to make predictions or compute other statistical quantities of interest in a graphical model.

Let's go back to one of the examples we considered before. Suppose we have data consisting of ratings, like or dislike, for 50 YouTube videos from everybody at a university. People are influenced through their network of friends. And since friendship is an undirected relationship we'll assume an undirected graphical model.

Remember that the graph structure in code the structure of statistical independence. Mainly, that each person rating is conditionally independent of everyone else's given the ratings of their friends. The Ising Model is appropriate since we have binary data. This is the canonical discrete graphical model. But everything we will discuss generalizes the other graphical models.

Each node in our graph, let's say there are P nodes for P people, has a variable associated with it. Node I has variable xI, which takes values minus 1 or plus 1. One data sample from the model corresponds to everyone's choice of like or dislike for a particular video and is, therefore, a binary vector of length equal to the number of people, P.

Now suppose that we've learned an Ising Model for people's preferences from the data. Beyonce releases a new video, and 15 people have rated the video. Other users haven't yet seen the video, and we want to predict each user's rating. This is useful, so that we can recommend the video to people who might like it.

We have information about 15 people, and each of these people is statistically dependent with their friends but there's a network effect where a nodes preference is dependent also with non neighbors due to propagational long paths and also effects can combine or add up. It's not at all obvious how to use the information we've observed in order to make predictions for other users, but that's exactly what the graphical model does.

It captures the statistical interaction between variables and tells us how to combine the pieces of information we've observed. From a statistical perspective, our goal is to compute the posterior distribution for a particular node, given our observations. This will be a number telling us how likely this node is to like or dislike the new Beyonce video conditional on the information we've observed It turns out that is enough for us to figure out how to compute the marginal distribution at a node in a using model, without conditioning at all.

The marginal, at a node, V, is just the probability distribution of the variable Xv. It is described by the probability that Xv is equal to plus 1 or minus 1. If the probability of Xv equals plus one is two-thirds, then the probability of Xv equals minus one is one-third.

Because the probabilities have to sum to 1. And this completely describes the distribution of Xv. Essentially, the marginal node v describes what you'd see if you only observed node v, and ignored everybody else. So despite there being a large network in the background we zoom in on one person.

We might ask what's the probability that Caroline will like the next YouTube video? In order to incorporate the observed information for some nodes, we can just fix the observed variables in the model to either like or dislike it turns out that we get a new easing model on the remaining nodes once we fix these variables.

And in this new model, a marginal end node describes the posterior probability we were after originally. So the basic problem we need to solve is computing the marginal end node, and the model where we haven't observed any of the nodes values. The reason why we need to do a computation to find the marginal at a single node, even though we haven't observed any of the other values, is that some nodes have a bias towards plus one or towards minus one and this propagates through the network.

5.3.6 Inference in Graphical Models - Part 2

Welcome back. In the last lecture, we discussed how computing the posterior distribution at a node, given observations of some node's values amounts to computing the marginal in a different model. So, our goal is to compute the marginal at a node. What exactly does it mean to compute a marginal of a node v?

Remember, that the marginal v just means the probability distribution of v if we just look at node v and ignore everyone else in the network. This intuition corresponds exactly to the mathematical operation of computing a marginal. We start with the joint probability distribution over the vector variables, X1 through Xp.

Let's suppose we're trying to find the probability that X1 = +1. This is obtained by summing the joint probability over all values of the other variables, X2 through Xp since we don't care what values those variables take. Doing such a summation is feasible for a small network of 20 nodes or so.

But with more nodes, we quickly run into trouble. This summation that we wrote down is over P minus 1 variables, each of which takes value plus 1 or minus 1. Meaning that there are 2 to the power P-1 terms in this summation, such an exponential dependence in P means that it's hopeless to actually compute this sum in most practical situations.

Looking at the numbers of terms and the sums suggest that it's difficult to compute marginals, but perhaps there's a smarter way to add things up. Unfortunately, computing marginals in general using models is computationally hard in a rather strong sense assuming widely believed conjectures in the theory of computational complexity are true.

This means that we can't hope to solve the problem for every single using model in any graph, but most real world scenarios are easier to deal with. Specifically, there are a variety of methods that work well to compute marginals or posteriors in many restricted classes and models that are useful in practice.

We'll get started by considering an important special case where the network has a path or chain structure. In probabilistic language, this is known as a Markov Chain. From the global Markov property for undirected graphical models, if you condition on an arbitrary node, the variables to the right are conditionally independent of the variables to the left since these two sets are separated by the node.

Let's suppose we want to compute the probability that node 1 is equal to plus 1. There's a remarkable way to do this that amounts to dynamic programming. Imagine for a second that we already know the marginal in node 2 or the probability that node 2 is plus 1.

In that case, we can just look at the two cases. Node 2 is plus 1 or -1 and compute the probability that node 1 is plus 1 in either case. This sort of local relationship is exactly the information contained in the model specification. The problem, of course is that we also don't know the marginal at node two.

The key is, it's enough to know the probability for X2 = +1 or minus 1 given the value of node 1. We can pretend that node two tells node one. If you're +1, then I'm +1 with probability 0.7 and -1 with probability 0.3. And if you're -1, then I'm +1 with probability 0.4 and -1 with probability 0.6.

We'll call this the message from node two to node one. How would node one use this information? Like you denote the probability of node 1 is +1. From the law of total probability, we can compute the probability that X2 is 1, which equals the probability that X2 is 1, given that X1 is 1 times the probability that X1 is 1 plus the probability that X2 is 1, given that X1 is -1 times the probability that X1 is -1, but this is equal to 0.7 times q+0.4(1-q).

We use this in a moment. So, just remember that we know the marginal at known two in terms of q. We also know from the graphical model specification, without doing any computations, the probability that X1 is +1, given that X2 is +1, suppose it's 0.2. Now by Bayes's rule, this is equal to the probability that X2 is +1, given that X1 is plus 1 times the probability that X1 is +1 divided by the probability that X2 is +1.

The first quantity on the right is part of node 2's message to node 1 and we assumed it was equal to 0.7. The probability that X1 is +1 is what we're interested in and we used q for this. The denominator, we solved for a minute ago and was 0.4+0.3 times q.

We can solve for q, which is the quantity we wanted. The probability that X1 is plus 1. We still haven't said, how node 2 can figure out this message to send to node 1. What we can say, though is that node p, which sits at the other end of the chain can send such a message to node p minus 1.

This is because for each value of Xp minus 1, node p can figure out the probability of being plus 1 or minus 1. It's at the end and there are no other nodes to affect it. By a similar computation to what we did for node 1 a moment ago, node p-1 can compute the message to send to node p-2, given the message from node p.

The general principle is the message from a node v along an edge can be computed once the incoming messages to v along all the other edges have been received. In this way, the messages are passed from one end of the chain to the other allowing us to compute the marginal at node one.

Of course, node one could send a message to node two similar to the one node p started with and the sequence of messages can go to the right eventually reaching node p. Each intermediate node can compute its marginal once it has received the appropriate messages from its two neighbors by using Bayes rule.

The total number of messages is twice the number of edges since each edge has a message going in either direction. So, the total computation time is linear in the number of nodes. This simple and efficient message passing algorithm is an instance of the belief propagation or some product algorithm.

We've seen that it can compute the marginals of each node. And therefore, the posterior probability at each node given partial observations of other nodes. Next time, we'll see how this generalizes beyond the chain graphs to tree structured graphical models as well as touch upon other inference algorithms.

5.3.6 Inference in Graphical Models - Part 3

Welcome back. In the last lecture, we saw how to efficiently computer marginals, or posteriors, on a chain graph using belief propagation, a simple message passing algorithm. In this lecture, we'll see how to apply the same reasoning to more general settings, as well as mention a few other approaches.

Our discussion about the chain graph revolved around each node passing messages to each of it's neighbors along the lines of if you are plus one then I am plus one with probability 0.7 and minus one with probability 0.3. And if you are minus one then I am plus one with probability 0.4 and minus one with probability 0.6.

Each node can compute the message to send out along an edge once messages from all the other edges have been received. This process starts at the ends of the chain since those nodes just have one adjacent edge and don't need to wait for any other messages. And then the messages are passed across the chain.

Each node can compute its marginal once it has all the messages from its neighbors. The first observation is that exactly the same belief propagation algorithm works if the graph is a tree structure instead of just being a chain. A tree is a graph with no cycles. Fundamentally, the algorithm needs to start with nodes of degree 1 which in our chain was just the ends of the chain, and in a tree these nodes are the leaves the belief propagation algorithm has appeared in various guises, initially in statistical physics and later in machine learning and artificial intelligence.

It provably computes the marginals for each of the nodes in a graphical model with a tree structure. Now, suppose that we want to find the single vector of likes and dislikes that is most probably for the new Beyonce video. It turns out that we can use a very similar message passing algorithm called max-product to compute the maximum probability assignment to the nodes.

This time the messages are of the form, if you're a plus 1 then, in the maximum probability assignment, I'm a minus 1. And if you're a minus 1, then in the maximum probability assignment I'm a plus 1. Of course, the messages can take all possible combinations. Just as before, if we want to find the maximum probability assignment taking account the observations for values of some of the nodes, we can find the maximum probability assignment in a new modified graphical model.

Belief propagation works on a tree. But let's think about what happens on a cycle. This is problematic because there's nowhere to start. None of the nodes have degree one, all of them have two neighbors. But even if we did have a place to start, for example we could attach a leaf to the cycle, the algorithm can run into trouble because the messages just go around and round the cycle.

So it's not completely clear what to do in a cycle. Let's think about the message passing algorithm we've discussed from the perspective of each known individually. Once messages from all neighbors except one have arrived, the message is computed and passed to the one remaining neighbor. In principal, these local update rules can be carried out on a graph that is not a tree.

Meaning it has cycles even if it's not guaranteed to give the right answer. Of course it's not clear how to initialize the values when there are cycles but this didn't stop people from trying to run an algorithm on graphs and cycles. You can just start things off with random messages and then run the algorithm for a while until it hopefully converges to something close to the correct answer.

Since cycles are also called loops, the belief propagation algorithm is often called Loopy Belief propagation when run on graph with cycles. It often works quite well in practice. There are various heuristics that are used to improve stability and performance of the algorithm such as adding damping factors to the messages.

Message passing algorithms form a useful class of algorithms because they run very fast and can often be implemented in a distributed manner. Some other types of algorithms for computations involving graphical models include Markov Chain Monte Carlo algorithms and variational methods. In the remainder of this lecture we'll briefly touch upon these algorithms.

Let's start with Markov Chain Monte Carlo called MCMC for short. Again, let's suppose that we want to compute the marginal in a node since computing a posterior probability, given partial observations, reduces to this problem. The idea behind MCMC is that if we can generate independent samples from the joint distribution, described by the graphical model, then we could easily estimate marginals, as well as other statistical quantities.

Remember that the marginal that a node v in an easy model is described by the probability that Xv is plus 1. Call this probability q. If you had samples from the entire joint distribution over p variables, you could estimate the value of q by throwing away all the variables except Xv from each sample.

Then we'd just be left with a bunch of samples of x of e and we could estimate q by the empirical average. This is just like estimating the bias of a coin by flipping it many times.

The outcome of coin toss is head.

So we wanna figure out how to produce samples from the joint distribution.

Each sample is a vector of length P of plus or minus ones. A markup chain describes evolution over time of this vector. Let's think about people logging into Facebook and deciding to make a post or not. People randomly log into Facebook one by one and if their friends make a post, they're more likely to read post or respond.

Of course there's randomness in this process since you might not post even if many of your friends did and conversely you may just post regardless if you've got something important to say. You might want to show off a picture surfing a big wave in Hawaii. But in any case, you post or not depending on the state of your friends, plus some randomness.

This simple mark up chain where each node updates, depending on the state of it's neighbors, is called the Gibbs Sampler or Glauber Dynamics. It turns out that after sufficiently many updates the joint distribution observed by taking a snapshot of the vector is the one we're looking for. We can just run such a mark up chain for awhile and get a sample.

Then we run it for awhile longer and get another sample. In this way we can produce as many samples as we want. The key question is for how many samples do we have to run the Markov Chain in order to get a sample that's close to the correct distribution?

This amount of time is known as the mixing time of the Markov chain, and depends on the structure of the network underlying the graphical model, as well as the nature of the interactions in the model. MCMC algorithms are very easy to implement, but often take a long time to run They're a good first thing to try, especially in complicated models.

Because they are sure to give an accurate answer, given enough time to run. So they work well in many situations, but can sometimes be too slow. Variational methods can be quite a bit faster. The basic idea is to approximate the true model by a much simpler one. And them perform computations on the simpler model.

One form of approximation, called Mean Field Variational Bayes, replaces the true distribution by one that is close in a distance called Kullback-Leibler Divergence. Let's call our simpler model, q. The idea is to solve a certain, fixed-point equation for q, and then compute in this simpler model. I won't be saying more about variational methods but it's good to know that they exist and provide sometimes faster alternative to MCMC.

In the last few lectures we've discussed several algorithmic approaches to compute predictions and other statistical quantities in a graphical model. In the next lecture we'll focus on a particular graphical model structure Has been tremendously useful in many applications, particularly involving temporal data. You may have heard of it before, it's called the hidden Markov model.

5.3.7 Hidden Markov Model (HMM)

Hi, we've talked about various aspects of graphical models including learning models from data and efficiently computing various statistical quantities. In this lecture, I'll describe a specific graphical model structure known as the Hidden Markov Model. And then in the next lecture, we'll discuss an inference algorithm called the Kalman Filter.

We'll make use of these concepts in the case study on GPS navigation. Hidden Markov models, or HMMs for short, have been fundamentally important in many applications involving speech, handwriting, text, music and bioinformatics. What do these applications have in common? Well most of them involve temporal or time-series data while in bioinformatics one often deals with genomic data that is by its nature sequential.

The basic idea behind HMMs is that there are some process, which you can think of as occurring over time, and we do not have access to it. Instead, we get only noisy observations of the process. This in itself is pretty general and can describe virtually any noisy data problem.

But in an HMM we make the additional assumption that the process is a Markov process. Let's be concrete and consider a simple example. Each day, I bike or drive to work. And my choice is influenced by the weather, which we'll assume is either sunny or rainy. You have data from the parking garage for when I park my car there over the past year.

And what you'd like to figure out is what the weather was for each day. A basic model for the weather would be to assume that the weather today depends on yesterday's weather. But conditional in yesterday's weather, today's weather is independent of previous days. This means that the weather, sunny or rainy, evolves according to a Markov chain.

It might be the case that if it's sunny today, it's sunny again tomorrow with probability 0.8 and rainy with probability 0.2. And if it's rainy today then it's sunny tomorrow with probability 0.8 and rainy with probability 0.2. So the weather stays the same with probability 0.8 and switches with probability 0.2.

In graphical model notation, this looks like a chain graph with a node for each day and the variable at each node for the weather on that day. We might use variable Xt to indicate the weather on day t. The Global Markov property implies that conditional on any node, the part of the chain to the right is independent of the part of the chain to the left.

Put another way, the future is independent of the past given the present. Now you don't observe the weather nodes, X1, X2, etc, and instead you observe whether I drove that day or not which we'll denote by Yt for day t. So to each node Xt in the Markov chain, we connect a new node, Yt whose variable indicates whether or not I drove to work that day.

That's it. This is a graph of a hidden Markov model. Notice that it is a tree graph since there are no cycles. One thing that's worth emphasizing is that in this graphical model, the edges indicate statistical relationships. In some of our earlier examples of graphical models, all of the nodes represented the same sort of data, such as like or dislike for a YouTube video, and edges represented interactions between people in a social network.

Here we have observed whether I biked or not, and this is a different sort of variable then the one saying whether or not it's sunny. But there is an edge because of the fact that I bike or not depending on the weather. HMMs have been used for decades in speech recognition.

Let's have a look to see how this model is used in that context. In a basic language model, each word in a sentence depends only on the preceding word, so the sequence of words forms a Markov chain. You can also have each word depend on the previous three words or n words, more generally.

And the same framework works there. We'll focus on the model where each word depends only on the preceding word. Of course, the computer or phone listening to your voice doesn't know the words you're saying. It just hears the sound of the words you produced. So these words, or waveforms, are the observed nodes.

To do speech recognition, the hidden Markov model is used to figure out the words that you're saying just from these observed nodes. The fact that we're using an HMM means that the graph structure is specified, but we still need to specify the Markov chain for the words. We have a sequence of words, X1, X2, X3, and so on, and maybe X1 is I and X2 is like.

The model then specifies the probability that X3 is the word cats, or any other word, according to model parameters which give the conditional probability of each possible word given the previous word. So for any time T, we have to specify the probability that XT is equal to some value WT, conditional on XT- 1 = WT- 1.

The remaining part of the model that we need to specify is the probability of observing Y1 given X1, Y2 given X2, Y3 given X3, and so on. This will indicate how does the word cats sound if X3 is the word cats. It's a noisy observation model. Our HMM for speech recognition is specified by the Markov transition probabilities between words as well as the observation model.

You can think of all of these as being described by a list of model parameters. The first fundamental question is how does one learn the parameters? The answer to this depends on the application. For language you can look at a large corpus of text documents and compute word transition frequencies to use in a model.

In other applications, including in computational biology, there is no way to directly observe the hidden variables. We only have access to the observational data. Even in this case, one can learn parameter values via a heuristic algorithm called the Baum-Welch algorithm which is just an instance of the expectation-maximization algorithm.

Let's now go back to our original motivating question and imagine that we have a known model and we want to use it to infer the most likely values of the hidden state variables. In our speech processing example, your phone listens to your voice and wants to figure out what are the words you've said.

Let's lump all the state, or hidden, variables into one vector, X, and the noisy observations into Y. We want to find a good guess for X, and the best guess is the maximum probability assignment to X given Y. Using Bayes' rule, we write $P(X|Y) = P(Y|X)P(X)/P(Y)$. We observe Y, so the probability of Y is some constant that does not depend on X.

This means that if we want to find the most likely X we can forget about Y and just find the X maximizing the probability of Y given X times the probability of X. The same basic message passing ideas we outlined for general geographical models apply here to allow efficiently computing these quantities.

In the context of HMMs, this goes by the name of the Viterbi algorithm. I'll mention a couple of other uses of HMMs. The exact same model can be used to describe product inventory in a store where I observe point of sale data, and the hidden variables indicate quantity of product in stock.

The number of bags of French roast coffee at the grocery store, for example, evolves over time according to a Markov chain. The grocery store may only have easy access to purchase data and restock data. Because of loss or theft, they don't actually know the number of bags of French roast sitting on the shelf.

They can use an HMM to estimate this number. HMMs have also been very useful for various text processing tasks including part of speech tagging. Here, you observe the words themselves and the hidden states indicate the part of speech of the word. You have a Markov model describing sentence structure.

If the last word was a noun, what's the likelihood that the current word is a noun, verb, adjective, preposition, and so forth. The reason why this is interesting is because many words can have multiple parts of speech with different meaning for each. Think about the word bear, as in grizzly bear, as compared to bear in the sentence, I can only bear listening to punk rock for at most five minutes.

Figuring out the part of the speech is an important first step in semantic understanding in text. Let's summarize. A hidden Markov model is a probabilistic model with latent or hidden variables describing the unobservable state of the system. This state evolves according to a Markov process. We then observe some noisy data that depends on the state of the system.

5.3.8 Kalman Filter

Welcome back. In this lecture, we are going to discuss the Kalman Filter which is an algorithm for estimating a process seen only through noisy observations. The Kalman Filter is simple and efficient and is closely related to the Hidden Markov Model. So, a lot of what we said in that context, applies here.

Some application domains include navigation and control of robots, vehicles, airplanes, and spacecraft. The Kalman Filter is also been used through data from sensor networks, econometrics, computer graphics, speech processing, and weather forecasting and many other areas. GPS navigation relies on the Kalman Filter and we will give more details on this topic in the case study.

It's going to be easiest to start with the Hidden Markov Model, like we discussed in the last lecture. Remember, that an HMM describes a system with state X sub T at time T, evolving over time according to a Markov process. The system is represented by an undirected graph with a chain structure with nodes X1 through XN.

The system state X sub T might be my heart rate at time T, at each time T, we observe a noisy version of XT which we denote by YT. So for each time T, we add a node we labeled YT with an edge connecting it to XT. If XT is my heart rate, then YT would be the electrical signal picked up by the heart rate monitor.

If we're designing a heart rate monitor, our goal is to get a good estimate of the state XT at time T given observation Y1 through YT. The problem is that the observation is a pretty noisy electrical signal. And using each observation alone leads to very noisy estimates. The graphical model framework, and specifically hidden Markov models, allow us to reason about how to combine data to perform inference tasks.

In discussing HMMs, we assumed the variables were discreet. But in many applications, it makes more sense to work with a continuum. The state, and observations, can also be vector valued. As would be the case if the state was my heart rate, and also my blood pressure. So, we'll assume that the variable X1 is a Gaussian random vector, and that each XT is obtained from XT -1, by multiplying by a linear operator or matrix A and adding some additional Gaussian noise, W sub T.

In general, A could change from one time to a next but the main point is that we're assuming you know A. These encapsulate the laws describing the evolution of the system. We model these XT variables as Gaussian to capture our uncertainty about them. The model described by the equation XT = A times XT-1 + WT is called the linear dynamical system.

Now we make noisy observations of XT by multiplying XT by matrix B and adding measurement noise Z sub T giving us YT = B x XT + ZT. These assumptions, that the variables are all jointly Gaussian, lead to the common filter for estimating XT given observations Y1 through YT.

The equations describing the filter itself don't require that the variables are Gaussian, but the filter is computing the correct posterior distribution only under assumption. So, what is the common filter doing at a high level? The filter specifies a way of combining noisy observations to produce an estimate of the hidden process.

There actually several ways of doing this. But one of the advantages of the common filter is that it works iteratively as observations are made and doesn't need to store any history only a best guess of the current time. This is in contrast to a batch algorithm that needs all of the observations up front.

Which just isn't possible in systems with data coming in over time. The filter can be thought of as working in two steps. At each time T, it produces an estimate of the process XT and then makes a correction to the estimate, given the noisy observation, YT. The correction is basically a weighted average of the estimate with the noisy observation.

The weighting in this average is specified by something called the gain of the filter. The gain of the filter is determining how much to emphasize the prediction of the state at time T based on previous observations versus the new observation at time T. If the gain is low, it means relatively less weight is placed the observation and the filter output will be smoother but less responsive to sudden changes.

If the gain is too high, the filter is sensitive to the observations and can be overly erratic. If we know the primaries of the Gaussian model described a moment ago, then optimal gain can be solve for exactly. Otherwise, you have to choose a value that balances moving and sensitivity.

Okay, so our goal is to compute the marginals for each of the XI variables, given observations Y1 through YT. A version of the forward, backward or I belif propagation algorithm can be used to this. Something we discussed in the context of hidden Markov models and naive implementation of belif propagation requires computing everything from scratch when we have a new observation YT plus 1.

The common filter is much faster. The two types of steps of the algorithm are predict steps and updates steps. At each time T, we're going to store the posterior probability density for the state XT given observations up to time T. Which you write as P of XT conditioned on observations

Y1 through YT.

This captures what we know about XT given the current observations. And bringing up probability density captures the inherent uncertainty. Under the Gaussian model described earlier, this is just a multivariate Gaussian distribution. While we could write everything out in terms of Gaussian densities, it's a lot cleaner to just keep things in terms of probability densities.

The prediction step, as the name implies, makes a prediction for the next state, XT plus 1, given the observations to time T. This will get us ready to use the next observation. Using base rule, we can write the conditional probability density for XT plus 1, given Y1 through YT as the integral over XT of P of XT plus 1, given XT times P(XT) given Y1 through YT.

The second factor is exactly the quantity we've already computed and the first factor P(XT + 1) given XT is something we can compute using the system model. Computing the integral will get the posterior for XT + 1 given only the information of the time T and that's the prediction step.

We'll now use what we computed in the prediction step for the update step to include the observation from time T plus 1. By base rule, the probability density of XT plus 1, given observations Y1 through YT plus 1 is proportional to the density of XT plus 1. Given observations Y1 through YT times the conditional density of YT plus 1, given TT plus 1.

The first factor is what we got from the predict step, and the second factor is just the probability density of observation based on the state. Which is described by the noisy observation model, which we assumed is known. So, the update step can be computed, and we've moved forward by one time unit.

You might be wondering, how did this whole process start? For T equals 1, the predict step is actually a lot easier. We're trying to predict X1 based on no observations at all, and this described by the prior distribution on X1 which is given by our model. That's it, we now have a good prediction at any time T for the users only past up to the current observations.

Of course, when all of the variables are jointly Gaussian, all of the conditional densities appearing in the predict and update steps are described by mean vectors and covariance matrices. And can be written succinctly using matrix algebra. These can then be used in a computer to compute estimates in real time.

It follows from theory that the common filter is optimal in case where the model perfectly matches the real system. And the covariance of the noise are exactly known. The conditions for optimality are rarely met in my real systems. But the filter works well in many applications because it is simple, and robust to inaccuracies and modeling assumptions.

This concludes the lecture on Kalman Filter. There are many generalizations of the filter, the non-linear systems, continuous time and more. The case study on GPS navigation goes into more detail on how the Kalman Filter is applied to solve an important problem relevant to all of us.

Case Study

Case Study 1: Identifying New Genes That Cause Autism

This lecture is a case study where we apply the concepts of network theory developed so far. This case study is about finding genes that cause autism. Autism is a complex disorder of brain development. It is characterized by impaired social interaction, communication and repetitive behavior. And if often comes with epilepsy.

Autism is highly heritable. It is important to determine the causative genes in order to cure the disorder, for example using gene therapies. Research related to determining genes that are associated to autism is a very active area and various causative genes have already been identified. However, the genetic basis of autism is still poorly understood, and only a fraction of cases can receive a genetic diagnosis.

Using network analysis methods, additional candidate genes can be determined and these genes can then be tested in the clinic. So what is the network on which we will build our analysis? Well, a lot of research nowadays is concentrated on building more and more accurate protein-protein interaction networks. In these networks, the nodes are genes, and an edge between gene A and gene B means that the protein produced by gene A interacts with the protein produced by gene B.

Protein-protein interaction networks are based on many different kinds of experiments. The problem with these networks is that there are a lot of false negative as well as false positive edges. First, researchers often focus on particular proteins that are known to be important for various diseases. While other proteins are not studied at all.

This leads to biased networks with many edges connecting the highly studied genes, but many missing edges among the genes that are yet to be discovered. Since it is difficult to determine protein interactions in our body, the data often comes from lab experiments directly on the proteins. However just noting that proteins can interact in the lab does not mean that they are indeed interacting in our body.

It is possible that although protein A and B can interact they never get to meet each other in our body. This explains the high rate of false positive edges. Hence a protein-protein interaction network should not be taken as the ground truth. Now, a list of genes known to cause autism is available through the gene autism database, SFARI.

We can highlight these genes in the protein-protein interaction network available, for example, through a database called BioGRID. But how do we now suggest other candidate genes that can then be experimentally validated with respect to their relationship with others? Before discussing further let's take a small detour and consider a completely different application where the equivalent problem arises.

So the Sicilian mafia, also known as La Cosa Nostra, is still quite active in Sicily and in fact, all of Italy. So let's assume we have access to a network of acquaintances of individuals in Italy. Similarly to the protein-protein network, the network of acquaintances is noisy. Data might have been collected from various sources.

One source could be Facebook, which has many false negative edges, since individuals who know each other might not be Facebook friends. On the other hand, the network of acquaintances could also contain links coming from cell phone data. However, just because an individual called someone related to the mafia, the phone call might not be related to any illegal activities.

Now various members of the Casa Nostra are well known. So we can highlight these members in the network of acquaintances. In this example, we have ten known members. But how are we now going to determine other suspects of the Cosa Nostra? It is reasonable to suspect that the members of the Cosa Nostra are connected to each other in the network of acquaintances.

Hence, it is reasonable to suspect individuals to be part of the mafia that would connect the already known members of the mafia in the network. So we are interested in central nodes in the network that lie on paths that connect the already known members of the mafia. Now on earlier lectures, we discussed how to find the shortest paths between any two nodes in the network.

Such paths are called geodesic paths. Now in this application we have multiple nodes. The known members of the Cosa Nostra. And we would like to find this molar sub graph that connects all these nodes. This problem is known as the minimum-weight Steiner tree problem. Although it is known to be np hard to find the minimum-weight Steiner tree.

There are good polynomial time algorithms that find approximate. Minimum-weight Steiner trees. Such approximations are based on algorithms for finding geodesic paths between nodes. We have a subset of nodes, in our case, the set of ten known mafulsy. First we compute the geodesic distance between all of these nodes in the subset.

Then we build a minimum spanning tree among these nodes. This is a tree of minimum weight, where the nodes are the people in this upset and the edges correspond to their geodesic distances in the original graph. This approach allows us to find approximate stiner trees that connect all known mafosian in the network of acquaintances.

We can extract the subnetwork from the original network. And we call this network the mafia interactome. Now we can perform a first test to make sure that the mafia interactome indeed makes sense. It is to be expected that the mafia is highly connected within the network of acquaintances.

We can check this by choosing subsets of ten nodes at random from the network of acquaintances. And building the interactum of these subsets. The connectivity of a network is captured, for instance, by the diameter of the network or the average geodesic distance between any two nodes in a network.

We can compute these network measures for all the randomly generated interactums and find that, indeed, the mafia network is more highly connected than any randomly chosen subnetwork. This analysis suggests that the mafia interactum indeed contains some useful information. Next we analysis the mafia interactum itself. The goal is to determine important nodes in this network.

In this application, it is reasonable to assume that a node is important if the central to the flow of information through the network. Assuming that information flows at a constant speed and take the shortest route in the network. Then the amount of information that flows through a node, is proportional to the number of geodesic paths that go through that node.

As we already discussed in earlier lectures, this is captured by the betweenness centrality hence the betweenness centrality is an appropriate centrality measure for this application. And we've already discussed how to compute it. Using the betweenness centrality, we reached the following conclusion. Mainly that these individuals here are suspects for belonging to the Cosa Nostra.

So after this detour through the world of mafulsy let's return to the problem of identifying new kind of gene for Autism. In did the problem is analogous to the problem of identifying suspects for the mafia. An underlying undirected network is available together with the list of know autism genes.

From these known autism genes, we can build an autism interactome. Just like in the mafia interactome, it is to be expected that the autism interactome is more highly connected than any randomly chosen sub network. This is the case since the genes causing autism are expected to be related by particular pathways.

Finally, the goal is to determine new candidate genes using the autism interactome. This problem translates into the problem of determining important genes in a network. Similarly, as in the mafia interactome, a gene is important because it is central to the information flow in the autism interactome. Problems in such genes are expected to cascade and lead to problems in the expression of many of the genes in their interactome, including the genes known to cause autism.

To summarize, we discussed how the network analysis tools that we introduced in this series of lectures can be used to determine candidate genes that cause autism, starting from a protein-protein interaction network, and a set of known autism genes. This problem is analogous to the problem of identifying suspects in the mafia given a network of acquaintances, and a set of known mafiosi.

While identifying mafia suspects is an important problem for international crime control. Identifying autism genes is crucial for developing therapeutic interventions against autism.

Case Study 2.1: GPS Navigation

Hi. In this lecture, we're going to do a case study. Focusing in on some of the details of GPS Navigation. GPS Navigation provides the perfect setting to illustrate the concepts we've discussed on interferons and graphical model, especially the common filter as used in practical systems that impact all of us everyday.

Part of what makes this problem challenging and interesting is that typically GPS data, which is acquired by satellites is supplemented with other navigation information from sensors in the vehicle itself, such as speed measurements. The Kalman filter is used to integrate the sources of information to give an accurate estimate.

GPS stands for Global Positioning System and was developed by the US Department of Defense. Since then, it has been made available for nonmilitary use. It's amazing how helpful GPS navigation is for getting from place to place when driving in a new city. I'll admit I'm probably more dependent on GPS than I ought to be.

Me, being absent minded, I'll sometimes enter in the directions to get to work on my phone just to be sure that I don't miss a turn. We all know more or less what GPS is. It's a system for telling you where you are. The idea is to give accurate position information for any number of devices, whether these are the phones in your pocket or planes in the air.

Location information is extremely useful and GPS is used in applications, including precise aircraft landing, mapping, both on land and ocean floor, vehicle navigation to coordinate transit systems and farming. So, there are many devices that need to figure out the position. You could imagine some sort of two way system where devices are receiving signals and also transmitting.

But for various reasons, GPS works by the device only receiving signals. For one, if everyone's device was transmitting information, the satellites would be overwhelmed with too many things to listen to. Even more problematic, GPS is supposed to work well anywhere on Earth. It would require a giant battery to have enough power to send signals up to the satellites all the time.

There are a bunch of GPS specific satellites in orbit. The system needs 24, but there are closer to 30 now. Each satellite periodically transmits a signal with its position and the time, and the GPS receiver then figures out the approximate distance to each of the satellites from which it gets a signal.

The approximate distance is figured out using the time difference between when the signal was sent and when it was received. Finally, with three or more distances from know satellite positions, it's possible to triangulate and solve for the position of an object on the surface of the

Earth at least four satellites will use most actual systems for various reasons.

Figuring out the distance from the GPS receiver to a satellite is called ranging and it depends on the accurate timing for propagation of signal. Since the signal travels at the speed of light, even tiny errors in the timing results in errors in the estimated distance. The clocks on the GPS receiver, such as the one in your phone just isn't as accurate as you might hope.

So, we get a noisy estimate of the distance to the satellite. When combining these signals, we get a noisy estimate of the position of the receiver at each time step. The accuracy is on the order of tens of meters. There are actually a bunch of other sources where there are noise.

These include small areas and estimates of the satellite position. Satellite clock errors. Errors to due to the multi path of the ranging signals. Noise and receiver electronics. Change in speed of propagation of the signal in the upper layers of the atmosphere and all this. We won't worry about the details too much and just lump together all of the errors.

Imagine that we were getting a noisy estimate of our location via GPS, as just described. Let's say, once per second to be concrete. Fundamentally, the GP location estimates are noisy and our application such as precise landing of a drone requires an accurate estimate of location. So, we want to get a higher degree of accuracy than the signal that we're getting.

Can we do something better than just using the GPS measurement at each time as our estimate? It helps to visualize what's going on with the location measurement. There's the true position of the drone, as it's flying around and the noisy measurements are bouncing around. The thing that saves the day is that the position of the drone changes smoothly from one time to the next.

So the GPS measurements are actually quite correlated and the collection of measurements over time can help to average out much of the noise, suppose for a moment that our drone hasn't moved. In this case, the true location is unknown but it's We collect a bunch of noisy measurements and we can average the measurements to get a much more accurate estimate of the position.

Now our drone is actually moving and this makes things a bit more complicated. This is where we're going to use the hidden markup model framework and the Kalman filter. In terms of the drone's position and velocity, the laws of physics give us equations of motion that predict where the drone will be at the next instance of time.

So if we have position and velocity is the state of the system, each of which is a three dimensional vector, the state of evolves over time according to a mark up process. Of course, there's noise involved since not everything is accounted for in the equations like fluctuating wind and this can be captured in the hidden mark up model.

I'm simplifying a bit by not including the controller inputs to the drone, but these can be subtracted out since they're known. We can think of the GPS location measurement as giving us a noisy observation of the state of the system at each time and the appropriate model is the hidden Markov model.

Our goal is to combine the observations up until the present time to produce an estimate of the state of the system. And for this task, the common filter we discussed is very well-suited, even better accuracy can be obtained by combining data from the vehicle or device itself with the GPS data.

For example, if we're thinking about a vehicle. It has an odometer, which measures how far it's gone over a given time period and it can also have a gyroscope, which measures orientation. If you know how far you've gone in one direction, then assuming you know where you started, you can figure out where you are, but these measurements are noisy too.

Tire pressure changes over time. Tires can slip or skid and the tires wear out, and these all cause the odometer reading to drift over time and no longer give an accurate estimate of the actual distance traveled. Similarly, gyroscopes will have bias due to changes in temperature. Combing measurements from these sensors together with GPS can be done using a common filter in a fairly direct way.

The dimension of the observation variables grows appropriately and the equations have to modified to account for new observation model. More extensive systems along the same lines go by the name Inertial Navigation Systems or INS for shorter. These include accelerometers and other sensors. Combining information from INS and GPS can get the best of both worlds.

The GPS signal can be obstructed. For example, by a forest, but INS still gives useful information. While the sensors in INS drift over time, but this can be corrected by incorporating the GPS measurements. This concludes our discussion on GPS navigation. It's quite remarkable how complex a hugely important system boils down to a relatively simple inference problem that can be solved using the methods we discussed.

The same graphical model inference framework, which has proved so useful in this and other engineering systems forms the foundation for how we think about much of inference and machine learning and it will continue to serve us a new and exciting applications. This also concludes the module on networks and graphical models.

We've learned about the important properties of networks, network models, statistical models, processes on networks, including graphical models, learning network structure from data and inference in graphical models. Looking to the future, many of the most exciting developments are happening in networks, including social networks and biological networks. We hope that you will find what you've learned to be useful and thank you for joining us.

Completing the Course & Post Course Notes (Philippe Rigollet and Devavrat Shah)

Parting Remarks from Course Co-Directors

Course Conclusion

Well, like all good things, this course has come to its end.

And we hope that you enjoyed taking it as much as we enjoyed teaching it.

Looking back, we have covered a large amount of material, ranging from making sense of unstructured data using clustering or principal component analysis, to making predictions using cutting-edge techniques, such as deep learning.

But above all, you have learned the phenomenal concepts behind data science, and hopefully demystified some data analysis techniques. You have gained valuable knowledge on how to use data to make better decisions.

Whether you found the material intuitive or challenging at times, you are now equipped with all you need to dive even deeper into the fascinating and growing world of data science.

On behalf of all the teaching team hear in MIT, we thank you for your attention and look forward to seeing you again in some more advance courses on data sciences. Good luck.

Thank you.