fast.ai: Intro to Machine Learning 1 (v1) (2018)

12 Video lesson transcripts based on youtube.com autogenerated subtitles.

Discussion forum: http://forums.fast.ai/.

Version 0.1 (2018-06-28) (proofreading progress: 0%).

If you'd like to contribute with proofreading, please see:
https://github.com/stas00/fastai-transcript/.

Credits: Jeremy Howard (fast.ai Teacher/Founder), Eric Perbos-Brinck (Video Timelines), Tim Lee (Outlines), Ottokar Tilk (Punctuator), Stas Bekman (Transcript making).

Table of Contents

Lesson 01

Outline

- Introductions and class basics

Video Timelines and Transcript

1. 00:02:14

- **AWS or Crestle Deep Learning**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Oh good, okay, so let me introduce everybody to everybody else. First of all, so we're here at the university of san francisco learning machine learning or you might be at home, watching this on video, so hey everybody wave here is the University of San Francisco graduate students. Thank you, everybody and wave back from the future and from home to all the students here. If, if you're watching this on youtube, please stop and instead go to course a I and watch it from there. Instead, that there's nothing wrong with YouTube, but I can't edit these videos after I've created them. So I need to be able to like, if you updated information about like what environments to use, how the technology changes, and so you need to go here right. So you can also watch the lessons from here. Here's lots of lessons and so forth right. So that's tip number one for the video tip number two for the video is because I can't edit them all. I can do is add these things called cards and cards or little things that appear in the top corner of the top right hand corner of the screen. So by the time this video comes out, I'm going to put a little card there right now for you to click on and try that out. Unfortunately, they're not easy to notice so keep an eye out for that, because that's going to be important updates to the video all right. So welcome we're going to be learning about machine learning today, then so after everybody in the class.

Here you all have Amazon Web Services setup, so you might want to go ahead and launch your AWS instance now or go ahead and create one short Jupiter notebook on your own computer. If you don't have Jupiter notebook setup, then what I recommend is you go to cresol, calm, wws or calm sign in there sign up , and you can then turn off enable GPU and click start. Jupiter and you'll have a Jupiter notebook instantly that costs you some money, it's three cents an hour. Okay, so if you don't mind spending three cents an hour to learn machine learning, here's a good way, so I'm going to go ahead and say start Jupiter, and so whatever technique you use there, you go one of the things that you'll find on the website is Links to lots of information about the costs and benefits and approaches to setting up lots of different environments for Jupiter notebook, both the deep learning and for regular machine learning, so check them out because there's lots of options. So if I then go open a Jupiter and Jupiter in a new tab Here I am in Crestle or on AWS or your own computer. We use the Anaconda Python distribution for basically everything you can install that yourself and again. There's

lots of information on the website about how to set that up we're also assuming that either you're using Crestle or there's something else which I really like called paper space comm, which is another place. You can fire up if you put a notebook pretty much instantly.

Both of these have already have all of the fastai stuff pre-installed for you. So as soon as you open up cresol or paper space, assuming you chose the paper space fastai template you'll see that there's a fastai, folder. Okay, if you are using your own computer or AWS, you'll, need to go to our github repo, fastai, fastai and clone it okay, and then you'll need to do a condor and update to install the libraries and again, that's all information we put on the Website and we've got some previous workshop videos to help you through all of those deaths so for this class, I'm assuming that you have a Jupiter notebook running okay, so here we are in the in the Jupiter notebook and if I click on fastai, that's what You get if you get clone or if you're in Chris all you can see our repo here. All of our lessons are inside the courses folder and the machine learning part one is in the ml one folder. If you're ever looking at my screen and wondering where are you look up here and you'll see that tells you the path fastai forces ml 1 and today we're going to be looking at less than one random forests. So here is lesson 1 RF you you

2. 00:05:14

- **lesson1-rf notebook Random Forests**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So there's a couple of different ways: you can do this, both here in person or on the video you can either attempt to follow along as you watch or you can just watch and then follow along later with the video it's up to you. I would maybe have a loose recommendation to say to watch now and follow along with the video later just because it's quite hard to motor tasks and if you're working on something you might miss a key piece of information which you're welcome to ask about. Okay. But if you follow along with the video afterwards, then you can pause, stop experiment and so forth. But anyway you can choose either way. I'm going to go of you, Topol header view, toggle tool bar and then full screen it so to get a bit more space. So the basic approach - we're going to be teaching here taking here, is to get straight into code, start building models not to look at theory. We've got to get to other theory, okay, but at the point where you deeply understand what it's for and at the point, that you're able to be an effective practitioner. So my hope is that you're going to spend your time focusing on experimenting. So if you take these notebooks and try different variations of what I show you try it with your own data sets the more coding you can do, the better. The more you'll learn.

Ok, don't in you know my suggestion, or at least, and all of my students have told me the ones who have gone away and spent time studying books of theory rather than coding found that they learnt less machine learning and that they often tell me they wish. That's one more time coding the stuff that we're showing in this course a lot of it's never been shown before. This is not a summary of other people's research. This is more a summary of 25 years of work that I've been doing in machine learning. So a lot of this is going to be shown for the first time and so that's kind of cool, because if you want to write a blog post about something that you learn here, you might be building something. But a lot of people find super useful. All right so there's a great opportunity to practice your technical writing and here's some examples of good technical writing. Okay, page by showing people stuff which you've it's not

like. Hey. I just learnt this thing. I bet you all know it often it'll be. I just want this thing and I'm going to tell you about it, and other people haven't seen it. In fact, this is the first course ever. That's been built on top of the fastai library, so even just stuff in the library is going to be new to like everybody, okay. So when we use a droopin, a notebook or anything else in python, we have to import the the libraries that we're going to use something. That's quite convenient as if you use these to auto reload commands at the top of your notebook.

You can go in and edit the source code of the modules and your notebook will automatically update with those new modules you won't have to like restart anything. So that's super handy, then, to show your plots inside the notebook you're wanting that plot in line. So these three lines appear at the top of all of my notebooks you'll notice, when I import the libraries that for anybody here who is a experienced Python programmer, I am doing something that would be widely considered very inappropriate. I'm importing star, okay, generally speaking in software engineering, we're taught to like it specifically figure out what we need and import those things the more experienced you are as Python programmer, the more extremely offensive practices you're going to see me use, for example, I don't follow. What's called pap 8, which is the normal style method style of code used in Python, so I'm going to mention a couple of things: first is go along with it for a while. Don't judge me just yet right, there's reasons that I do these things and if it really bothers, you then feel free to to change it right. But the basic idea is data. Science is not software engineering right, there's a lot of overlap. You know we're using the same languages and in the end, these things or may become software engineering projects. But what we're doing right now is we're prototyping models and prototyping models has a very different set of best practices that are taught basically nowhere right, they're, not really even really written down, but the key is to be able to do things very interactively and very iteratively.

Right so, for example, from library import, star means you don't have to figure out ahead of time what you're going to need from that library. It's it's all there. Okay, also because

3. 00:10:14

- **`?display` documentation, `??display` source code**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

We're in this wonderful, interactive, Jupiter, environment, it lets us understand, what's in the libraries really well so, for example, later on, I'm using a function called display right. So an obvious question is like well, what is display, so you can just type the name of a function and press shift enter member shift enter is is to run a cell and it will tell you where it's from right. So anytime, you see a function. You are not familiar with. You can find out where it's wrong and then, if you want to find out what it does put a question mark at the start. Okay and here you have the documentation and then particularly helpful for the faster I library, so the faster. I library I try to make as many functions as possible, be like no more than about five lines of code. It's just going to be really easy to read right. If you put a second question mark at the star, it shows you the source code of the function right, so all the documentation plus the source code. So you can see like nothing has to be mysterious and we're going to be using the other library we'll use. A lot is scikit-learn, which is kind of implements a lot of machine learning stuff in Python. The scikit-learn source code is often pretty readable and so very often, if I want to really understand something I'll just go question mark question mark and

the name of the scikit-learn function, I'm typing and I'll just go ahead and read the source code.

As I say, the first day, I library in particular is designed to have source code, that's very easy to read and we're going to be reading it a lot. Okay, all right. So today we're going to be working on a cattle competition called Blue Book for bulldozers. So the first thing we need is

4. 00:12:14

- **Blue Book for Bulldozers Kaggle competition: predict auction sale price,**
- **Download Kaggle data to AWS using a nice trick with FireFox javascript console, getting a full cURL link,**
- **Using Jupyter "New Terminal"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

To get that data, so if you go Kaggle bulldozers, then you can find it so cackle competitions allow you to download a real-world data set. That's somebody a real problem that somebody's trying to solve and solve it according to a specification that that actual person with that actual problem decided would be actually helpful to them right. So these are pretty authentic experiences for applied machine learning. Now, of course, you're missing. All the bit that went before, which was why did this company to start up the side that predicting the option sale price of bulldozers was important. Where did they get the data from? How did they clean the data and so forth? Okay and that's all important stuff as well, but the focus of this course is really on what happens next, which is like how do you actually build the model? One of the great things about you working on Kaggle, competitions, whether they be running now or whether they be old ones, is that you can submit yours to the leaderboard, even old, closed competitions. You can submit to the leaderboard and find out. How would you have gone right and there's really no other way in the world of knowing whether you're competent at this kind of data in this kind of model than doing that right? Because otherwise, if your accuracy is really bad? Is it because this is just very hard like it's just not possible, then the data is so noisy.

You can't do better, or is it actually that it's an easy data set and you've made a mistake and like when you finish this course and apply this to your own projects. This is going to be something you're going to find very hard and there isn't a simple solution to it, which is you're now using something that hasn't been on cog or your own data set. Do you have a good enough answer or not? Okay, so we'll talk about that more during the course and in the end we just have to know that we have good effective techniques to reliably building baseline models. Otherwise, yeah there's really no way to know, there's no way other than creating a cackle competition or getting you know a hundred top data scientists to work at your problem to really know what's possible. Socal competitions are fantastic for for learning and, as I've said many times, I've learned more from kept from competing in cackled competitions and everything else. I've done in my life so to compete in the caracal competition. You need the data, this one's a an old competition. So it's not running now, but we can still access everything. So we first of all want to understand what the goal is, and I suggest that you read this later, but basically we're to try and predict the sale price of heavy equipment and one of the nice things about this competition is that, if you are like me, You probably don't know very much about heavy heavy industrial equipment options right.

I actually know more than I used to because my toddler loves building equipment, so we actually like watched, youtube videos about front end, loaders and forklifts, but you know two months ago I was, you know a real layman, so one of the nice things is that machine Learning should help us understand a data set not just make predictions about it, so by picking an area which we're not familiar with it's a good test of whether we can build an understanding right because otherwise, what can happen is that your intuition about the data can Make it very difficult for you to be open-minded enough to see what does the data really say? It's easy enough to download the computer sorry to download the data to your computer. You just have to click on the data set. So here is train, zip and click download right, and so you can go ahead and do that if you're running on your own computer right now, if you're running on AWS, it's a little bit harder right, because unless you're familiar with textmode browsers like a links or Links, it's quite tricky to get the data set to cable, so a couple of options: one is you could download it to your computer and then SCP it to AWS, so SCP works just like SSH, but it copies data rather logging in I'll show you a trick, Though that I really like and it relies on using Firefox for some reason - chrome - doesn't work correctly with cable for this. So if I go on Firefox to the website, eventually you and what we're going to do is we're going to use something called the JavaScript console. So every web browser comes with a set of tools for web developers to help them see what's going on, and you can hit 0 3 here developer, control-shift a okay, so you can hit ctrl shift.

I to bring up this. This web developer tools and one of the tabs is network okay, and so then, if I click on train zip and I click on download, okay and I'm not even going to download on let's gon na, say, cancel but you'll see down here. It's shown me all of the network connections that were just initiated right and so here's one which is downloading a zip file from storage. Google API is calm, blah blah blah. That's probably what I want now that looks good, so what you can do is you can right-click on that and say copy copy as curl, so curl is a UNIX command like wget that downloads stuff right. So if I go copy as curl that's going to create a command that has all of my cookies headers everything in it necessary to download this authenticated data set. So if I now go into my server right and if I paste that you can see a really really long, curl command, one thing I notice is that at least recent versions have started adding this 2.0 thing to the command that doesn't seem to work with all Versions of curl, so something you might want to do, is to oopsy-daisy a PE is to pop that into an editor, find that to get rid of it and then use that instead, okay, now one thing to be very careful about by default, curl downloads, the file And displays it in your terminal. So if I try to display this, it's going to display gigabytes of binary data in my terminal and crash it okay.

So to say that I want to output it using some different file name, I always type o for output file. Name and then the name of the file bulldozers dot and make sure you give it a suitable, a suitable extension. So in this case the file was train's, it okay, so bulldozers dot, zip there. It is okay, and so there it all is so I could make directory bulldozers and I couldn't move my zip file into there oops wrong way around . Yes, thank you! Ah, you you, okay and then you, if you don't have unzip install, you may need to sudo apt, install unzip or, if then you're, on a Mac that would be brew. Install unzip! If brew doesn't work, you haven't, got homebrew installed, so make sure you install it and then unzip, okay, and so there the basic steps. One nice thing is that if you're using Crestle most of the data sets should already be pre-installed for you. So what I can do here is, I can say, open a new tab. Here's a cool trick in Jupiter. You can actually say new terminal and you can actually get a web-based terminal and so you'll find one cresol. There's a slash. Data sets folder slash, data sets flash Kaggle, slash, data set, slash birthday. I often the things you need are going to be in one of those places. Okay, so assuming that we don't have it already downloaded in paper actually paper, space should have

most of them as well.

Then we would need to go to far say I let's go into the courses, machine learning, folder and what I tend to do is I tend to put all of my data for a course into a folder called data. You'll find that if you try and if you're using what we using get right, you'll find that that doesn't get added to get because it's in the get ignore right. So so don't worry about creating the data folder! It's not going to screw anything up. So I generally make a folder called data, and then I tend to create folders for everything I need there. So in this case I'll make bulldozes CD and remember. The last word of the last command is exclamation: mark dollar, I'll, go ahead and grab that kill command again: , you, okay and zip bulldozers. There we go okay, so you can now see. I generally have like anything that would change that. Might change from person to person I kind of put in a constant, so here I just defined something called path, but if you've used the same path, I just did just got to go ahead and run that and let's go ahead and keep moving along. So we've now got all of our libraries imported and we've said the path to the data. You can run.

5. 00:23:55

- **using `!ls {PATH}` in Jupyter Notebook**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Shell commands from within Tripta notebook by using an exclamation mark. So if I want to check what's inside that path, I can go. Ls data slash bulldozers, okay, and you can see that works or you can even use Python variables. If you use a Python variable inside a Jupiter show command, you have to put it in curlies, okay, so that makes me feel good that my path is pointing at the right place, if you say LS, curly capital's path and you get nothing at all, then you're Pointing at the wrong spot - yes, this up here, usually yeah, so the curly brackets refer to the fact that I put an exclamation mark at the front, which means the rest of this is not a Python command. It's a bashed, command and bash doesn't know about capital path, because capital Park is part of them. So this is a special Jupiter thing which says: expand this Python thing please before you pass it to the show question. Thank you. So the goal here is to use the training set, which contains data through the end of 2011 to predict the sale price of bulldozers, and so the main thing to start with then is of course, to look at the data. Now the data is in CSV format. Right so one easy way to look at the data would be to use shell command head to look at the first two lines: paired bulldozers and even tab-completion works here.

Jupiter does everything right, so here's the first few five lines? Okay, so there's like a bunch of column, headers and then there's a bunch of data, so that's pretty hard to look at. So what we want to do is take this and read it into a nice tabular format. Okay, so just Terrance putting classes on mean. I should make this bigger, or is it okay? Is this big enough font size? Okay? So this kind of data where you've got columns representing a

6. 00:26:14

- **Structured Data vs data like Computer Vision, NLP, Audio,**
- **'vimimports.py'in/fastai, 'low_memory=False', 'parse_dates',**
- **Python 3.6 format string f'{PATH}Train.csv',**

- **'display_all()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Wide range of different types of things, such as an identifier of value, a currency, a date, a size. I refer to this as structured data. Now I say I refer to this as structured data because, like there have been many arguments in the machine learning community on Twitter, about what is structured data weirdly enough. This is like the most important type of distinction is between data. That looks like this and data. Like images where every column is of the same type, like that's the most important distinction in machine learning, yet we don't have standard accepted terms. So I'm going to use the term structured and unstructured, but note that other people you talk to, particularly in NLP and NLP people, use structured to mean something totally different right. So when I refer to structured data, I mean columns of data that can have varying different types of data in them by far the most important tool in Python. If you're working with structure data is pandas, pandas is so important that it's one of the few libraries that everybody uses the same abbreviation for it, which is PD so you'll, find that one of the things I've got here is from fast. Ai imports import star. Okay, the faster I imports module has nothing but imports of a bunch of hopefully useful tools. So all of the code for fastai is inside the fast a I directory inside the fastai repo, and so you can have a look at imports and you'll see.

It's just literally a list of inputs and you're fine, there pandas as PD, and so everybody does this right. So you'll see lots of people using PD dot, something they're always talking about pandas. So pandas lets us read a CSV file, and so when we read the CSV file, we just tell it the path to the CSV file, a list of any columns that contain dates - and I always add this low-memory - equals false. That's going to actually make it read more of the file to decide what the types are. This here is something called a Python, 3.6 format string. It's one of the coolest parts of python 3.6. You've probably used lots of different ways in the past in Python of interpolating variables into your strings. Python 3.6 has a very simple way that you'll probably always want to use from now on, and it's you to create a normal string. You type an F for the start and then, if I define a variable, then I can say hello: curlies function, okay. This is kind of confusing. These are not the same curlies that we saw earlier on in that LS command right. That LS command is specific to Jupiter and interpolates code into shell code. These curlies are Python 3.6 format string curlies. They require an F at the start. So if I get rid of the F, it doesn't interpolate. Okay, so the F tells it to interpolate, and the cool thing is inside that curlies.

You can write any Python code you'd like just about so, for example, name dot, Papa hello, Jeremy, okay, so I use this all the time and it doesn't matter because it's a format string, it doesn't matter. If the thing was. I always forget my age. I think I'm 43, it doesn't matter if it's an integer right normally, if you like to do string concatenation with integers place and complains no such problem here, okay, so so this is going to read path, slash, train dot, CSV into a thing called a data frame. Pandas data frames and hours data frames are kind of pretty similar, so if you've used R before then you'll find that this is a you know reasonably comfortable. So this file is nine point. Three Meg and it's size is sorry: 112 Meg, 112 Nick and it has 400,000 rows in it. Okay, so it takes a moment to import it, but what it's done? We can type the name of the data frame, DF raw and then use various methods on it. So, for example, deer fur or tail will show us the last few rows of the data frame by default. It's going to show the columns along the top and the rows down the side, but in this case there's a lot of

columns. So I've just said dot transpose to show it the other way around. I've created one extra function here display all normally, if you just type DF R or if it's too big, to show conveniently it truncates it and put little ipsus in the middle.

So the details, don't matter, but this is just changing your couple of settings to say even if it's got a thousand rows in a thousand columns, please still show the whole thing. Okay, so this is finished. I can actually show you that. So if I just type this is really cool in in Jupiter notebook, you can type a variable with almost any kind a video HTML, an image whatever and it'll generally figure out a way of displaying it for you, okay. So in this case it's a pandas data frame. It picks it out a way of just playing it for me, and so you can see here that by default it's actually doesn't show me the whole thing. So so here's the data set we've got a few different rows. This is the last bit the tail of it alright last few rows. This is the thing we want to predict price, okay and then all of the other. We call this the dependent variable, the dependent variable is the price, and then we got a whole bunch of things we could predict it with and when I start with a data set, I tend yes Terrance. How can I give you this hello, Jeremy, hi, Tara? I've read in books that you should never look at the data.

7. 00:33:14

- **Why Jeremy's doesn't do a lot of EDA,**
- **Bulldozer RMSLE difference between the log of prices**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Because of the risk of overfit, why do you start by looking at the data yeah, so I think she's gon na mention I actually kind of don't like I. I want to find out at least enough to know that I've like managed to imported okay, but I tend not to really study it at all at this point, because I don't want to make too many assumptions about it, I would actually say most books say the Opposite most books do a whole lot of expediate exploratory data analysis. First yeah academic books say that that's one of the biggest risks of everything but the practical books say: let's do some EDA first yeah, so that the truth is kind of somewhere in between, and I generally I generally try to do machine learning, driven EDA and that's What we're going to learn today? Okay, so the do thing I do care about, though, is what's the purpose of the project and for capital projects. The purpose is very easy. We can just look and find out, there's always an evaluation section. How is it evaluated - and this is evaluated on root - mean squared log error, so this means they're going to look at the difference between the log of our prediction of price and the log of the actual price and then they're going to square it and addemup. Okay, so because they're going to be focusing on the difference of the logs, that means that we should focus on the logs as well, and this is pretty common like for a price generally, you care not so much about.

Did I miss by ten dollars, but did I miss by ten percent right? So if it was a million dollar thing and you're a hundred thousand dollars off or if you're, it's a ten thousand dollar thing and you're a thousand dollars off often we would consider those equivalent scale issues and so for this oxygen problem, the organizers are telling us They care about ratios more than differences, and so the log is the thing we care about. So the first thing I do is to take the log okay. Now NP is numpy, I'm assuming that you have some familiarity with numpy. If you don't, we've got a video called deep learning workshop, which actually isn't just for deep learning. It's Rahal! It's basically for this as well, and one of the parts there, which we've got a

time coded link to there's a quick introduction to numb pay, but basically numpy lets us treat arrays matrices, vectors, high dimensional chances as if they're Python variables, and we can do stuff. Like log to them and it'll apply it to everything, numpy and pandas work together very nicely. So, in this case, DF fraud, sale price is pulling a column out of a pandas data frame which gives us a pandas series right shows us the sale prices and their indexes right and a series can be passed.

8. 00:36:14

- **Intro to Random Forests, in general doesn't overfit, no need to setup a validation set.**
- **The Silly Concepts of Cursive Dimensionality and No Free Lunch theorem,**
- **Brief history of ML and lots of theory vs practice in the 90's.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

To a number I function, okay, which is pretty handy, and so you can see here. This is how I can replace a column with a new column, pretty easy. So, okay, now that we've replaced its sale price with its log, we can go ahead and try to create a random forest. What's a random forest we'll find out in detail, but in brief, a random forest is a kind of universal machine learning technique. It's a way of predicting something that can be of any kind. It could be a category like. Is it a dog or a cat, or it could be a continuous function? You aspera like price. It can predict it with columns of pretty much any kind. Pixel data zip codes, revenues whatever in general, it doesn't overfit it can and will learn to check whether it is, but it doesn't generally overfit too badly and it's very very easy to make to stop it from overfitting. You don't need, and we'll talk more about this - you don't need a separate validation set in general. It can tell you how well it generalizes, even if you only have one data set, it has few. If any statistical assumptions it doesn't assume that your data is normally distributed, it doesn't assume that the relationship so linear it doesn't assume that you've just specified the interactions. It requires very few pieces of feature engineering for many different types of situation.

You don't have to take the log of the data you don't have to multiply interactions together, so in other words, it's a great place to start right. If your first random forest does very little useful, then that's a sign that there might be problems with your data. Like it's designed to work pretty much first off, can you please throw it out or towards this gentleman? Thank you. What about the national year and, of course, yeah great question, so there's this concept of curse of dimensionality. In fact, there's two concepts: I'll touch on curse of dimensionality and the low free lunch theorem. These are two concepts. You often hear a lot about, they're, both largely meaningless and basically stupid, and yet I would say, maybe the majority of people in the field. Not only don't know that, but think the opposite, so it's well worth explaining the curse of dimensionality. Is this idea that the more columns you have it basically creates a space, that's more and more empty, and this is kind of fascinating mathematical idea, which is the more dimensions you have. The more all of the points sit on the edge of that space. Alright. So if you've just got a single dimension, where things are like random, then they're spread out all over right. Where else, if it's a square, then the probability that they're in the middle means that they've kind of been on the edge of either dimension. So it's a little bit less likely that they're not on the edge edge dimension.

You add, it becomes more addictive, ly, less likely that the point isn't on the edge of at least one dimension right and so basically in high dimensions. Everything sits on the edge, and what that means in theory is that the distance between points is much less meaningful. And so, if we assume that somehow that matters that it would suggest that when you've got lots of columns - and you just use them without being very careful to remove the ones, you don't care about that, somehow things won't work. That turns out just not to be the case. It's not the case for a number of reasons. One is that the points still do have different distances away from each other, just because they're on the edge they still do, vary and far where they are from each other, and so this point is more similar at this point that it is to that point. So even things will learn about K, nearest neighbors, actually, work really well really really well in high dimensions, despite what the theoreticians claimed and what really happened here was that in the 90s theory totally took over machine learning, and so particularly there was this concept of these Things called support, vector machines that were theoretically very well justified, extremely easy to analyze mathematically and you could like kind of prove things about them and we kind of lost a decade of real practical development.

In my opinion, and all these theories became very popular, like the curse of dimensionality nowadays and a lot of theoreticians hate this and the the world of machine learning has become very empirical, which is like which techniques actually work, and it turns out that, in practice, building Models on lots and lots of columns works really really well so yeah. The other thing to quickly mention is the no free lunch theorem, there's a mathematical theorem by that name that you will often hear about their claims that there is no type of model that works. Well, for any kind of data set, which is true and is obviously true, if you think about it in the mathematical sense, any random data set by definition is random right. So there isn't going to be some way of looking at every possible random data set. That's in some way more useful than any other approach in the real world. We look at data which is not random mathematically. We would say it sits on some lower dimensional manifold. It was created by some kind of caused all structure. There are some relationships in there. So the truth is that we're not using random datasets, and so the truth is in the real world. There are actually techniques that work much better than other techniques for nearly all of the datasets you look at, and nowadays there are empirical researchers who spend a lot of time studying this, which is which techniques work a lot of the time and ensembles of decision trees.

With which random for a one is perhaps the technique which most often comes at the top, and that is despite the fact that until the library that we're showing you today first day, I came along, there wasn't really any standard way to pre-process them properly and to Properly set their parameters, so I think it's even more strong than that so yeah. I think this is where the difference between theory and practice is is, is huge. So when I try to create a ratso random forest regressor, what is that random forest regressor? Okay? It's part of something called scikit-learn. scikit-learn is scikit-learn. It is by far the most popular and important package for machine learning in python. It does nearly everything it's not the best at

9. 00:43:14

- **RandomForestRegressor, RandomForestClassifier**
- **Stack Trace: how to fix an error**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Nearly everything, but it's perfectly good at nearly everything so like you - might find in the next part of this course with your net you're, going to look at a different kind of decision. Tree ensemble called gradient roost in trees, where actually there's something called x3 boost, which is better than gradient, boosting trees in psyche alone. But it's pretty good at everything. So, where I'm really going to focus on cycle in random forests, you can do two kinds of things with a random forest. If I hit tab, I haven't imported it. So let's go back to where we import . So you can hit tab in Jupiter. Notebook to get tab-completion for anything, that's in your environment, you'll see that there's also a random forest classifier. So in general, there's an important distinction between things which can predict continuous variables. That's called regression and therefore a method for doing that would be a regresar and things that predict categorical variables, and that is called classification and the things that do that are called plasa fires now. So in our case, we're trying to predict a continuous variable pres. So therefore, we are doing regression and therefore we need a regress or a lot of people incorrectly use the word regression to refer to linear regression. Now it is just not at all true or appropriate. Regression means an assumed learning model. That's trying to predict some kind of continuous outcome.

It has a continuous dependent variable, so pretty much everything in scikit-learn has the same form. You first of all create an instance of an object for the machine learning model you want. You then call fit passing in the independent variables, the things you're gon na use to predict and the dependent variable the thing that you want to predict. So, in our case, the dependent variable is, is the data frames, sale, price column, and so we, the thing we want to use to predict is everything except that in pandas, the drop method returns. A new data frame with a list of columns removed right. Well, a list of rows or columns removed, so access equals 1 means removed columns. So this here is the data frame containing everything except for sale, price. Okay, so you just pass a list. Let's find out so to find out. I could hit shift tab and that will bring up the you know a quick inspection of the parameters in this case. It doesn't quite tell me what I want. So if I hit shift tab twice, it gives me a bit more information. Yes - and that tells me it's a single label or list like list like means, like anything you can index in Python, there's lots of things by the way. If I hit three times, it will give me a whole little window at the bottom. Okay, so that was shift tab. Another way of doing that, of course, which we learned would be question mark question mark DF, bra drop.

Okay, sorry question mark question mark would be the source code for it, for a single question mark is the documentation. So I think that trick of like tab, complete shift-tab parameters, question mark and double question mark for the docs and the source code like if you know nothing else about using Python libraries, know that, because now you know how to find out everything else. Okay, so we try to run it and it doesn't work okay. So why didn't it work? So anytime? You get a stack trace like this. So an error. The trick is to go to the bottom, because the bottom tells you what went wrong a buffer. It tells you all of the functions the court other function could cause other functions to get. There could not convert string to float conventional, so there was a column name. Sorry a there was a value rather inside my data set conventional. The word conventional and it didn't know how to create a model using that string. Now, that's true. We have to pass numbers to most machine learning models and certainly to random forests, so step one is to convert everything into numbers, so our data set contains both continuous variables so numbers where the meaning is numeric like price, and it contains categorical

10. 00:48:14

- **Continuous and categorical variables, `add_datepart()`**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Variables which could either be numbers where the meaning is not continuous, like zip code, or it could be a string like large, small and medium. It's a categorical and continuous variables. We want to basically get to a point where we have a data set where we can use all of these variables, so they have to all be numeric and they have to be usable in some way. So one issue is that we've got something called say all date, which you might remember right at the top. We told it that that's a date, so it's been passed as a date, and so you can see here it's Dana type, D, type, very important thing. Data type is date/time 64-bit, so that's not a number right, and this is actually where we need to do our first piece of feature engineering right inside a date. There's a lot of interesting all right. So since you've got the catch box, can you tell me what are some of the interesting bits of information inside a date? What we can see like a time series? That's true! I'm hadn't expressed for real. What are some columns that we could pull out of this yeah month? The date as in like it come Ian, at least to be a number yeah month quarter, you're a pleasure to your right and get some more behind you just pass it to your right. You go, you got some more columns for us the day of month. Yeah keep going to the right yeah week, 30 of week, yeah week of here, yeah, okay, I'll, give you a few more like that you might want to think about would be like.

Is it a holiday? Is it a weekend? Was it raining that day? Was there a sports event that day like it depends a bit on what you're doing right so like if you're predicting soda sales in soma, you would probably want to know? Was there a San Francisco Giants ball game on that day, right so like what's in a date, is one of the most important pieces of feature engineering you can do and no machine learning algorithm can tell you whether the Giants were playing that day and that it Was important right, so this is where you need to do feature engineering, so I to as much things as many things automatically as I can for you right so here I've got something called add date pad. What is that? It's something inside fastai, dot, structured, okay, and what is it? Well, let's read the source code here it is so you'll find. Most of my functions are less than half a page of code. Alright, so here is something it's going to so, rather than often rather than having Docs I'm going to try to add Doc's over time, but that is their design. You can understand them. I reading the code so we're passing in a data frame and the name of some field, okay, which in this case was sale date, and so in this case we can't go D, F, dot field name because that would actually find a field called field name. It literally so DF square bracket field name, is how we grab a column where that column name is stored in this variable.

Okay, so we've now got the field itself, the series yeah, and so what we're going to do is we're going to go through. All of these different strings right - and this is a piece of Python which actually looks inside an object and finds a attribute with that name. So this is going to go through and you can again you can google for Pais and get attribute it's a cool. Little advanced technique, but this is going to go through it's going to find for this field. It's going to find its Year attribute now. Planter's has got this interesting idea, which is, if I actually look inside, let's go field equals. This is the kind of experiment I want you to do right, play around say all date: okay, so I've now got that in a field object, and so I can go field right and I can go field dot, tab, okay and, let's see is year in there. Oh, it's not okay! Why not! Well that's, because year is only going to apply to pandas series that date time objects. So what pandas does is it lets out different methods inside attributes that are specific to what they are so date/time

objects will have a DT attribute defined and at that is where you'll find all the date/time specific stuff. So what I went through was, I went through all of these and picked out all of the ones that could ever be interesting for having any reason right - and this is like the opposite of the curse of dimensionality - it's like if there is any column or any Variant of that column that could be ever be interesting at all.

Add that to your data set and every variation of it, you can think of there's no harm in adding more columns nearly all the time right. So in this case, we're going to go ahead and all of these different attributes and so for every one I'm going to create a new field. That's going to be called the name of your field with the word date removed, sort of a sale and then the name of the attribute, so we're going to get a sale year, sale months so a week say all day, etc, etc. Okay and then at the very end, I'm going to remove the original field right because remember, we can't use, say all date directly, because it's not a number, so your sickness only worked because it was a date type. Did you make the data it was already saved as one in the original yeah, it's already a date type and the reason it was a date tonight is because when we imported it we said PA's dates, equals and told pandas, it's a date type. So as long as it looks date-ish and we tell it to parse it as a date, if you don't have an intranet, i I think there might be, but for some reason it wasn't ideal like. Maybe it took lots of time or it didn't always work or for some reason I had to list it here. I would suggest checking out the docs for pandas, don't read CSV and maybe on the forum. You can tell us what you find, because I can't remember offhand you I got telephoning so how about the time zone.

Let's do that one on the same forum thread that savanah creates, because I think it's a reasonably advanced question, but, generally speaking the time zone in a properly formatted date will be included in the string and it should format it. It should pull it out correctly and turn it into a universal time zone. So, generally speaking, it should handle it for you, so I noticed you for indexing a column to shrink when we use the is there any consideration? The square brackets one is safer, particularly if you're assigning to a column. If it didn't already exist, you need to use the square brackets format. Otherwise, you'll get weird errors, so the square brackets format is sofa. The dart version saves me like a couple of keystrokes, so I probably use it more than I should in this particular case, because I wanted to grab something that was had field name was had something inside it wasn't the name itself. I have to use square brackets, so square brackets is going to be your your safe bet, if in doubt so after I run that you'll notice that DF r or dark columns gives me a list of all of the columns just as strings and at the end There they all are right, so it's removed sale date and it's added all those. So that's not quite enough. The other problem is that we've got a whole bunch of strings in there right. So you can just think that they're doing to pass a bet so is like low high medium. Thank you.

So

11. 00:57:14

- **Dealing with strings in data (“low, medium, high” etc.), which must be converted into numeric coding, with train_cats() creating a mapping of integers to the strings.**
- **Warning: make sure to use the same mapping string-numbers in Training and Test sets,**
- **Use “apply_cats” for that,**
- **Change order of index of .cat.categories with .cat.set_categories.**

A pandas actually has a concept with a category data type, but by default it doesn't turn anything into a category for you. So I've created something called Train cats, which creates categorical variables for everything. That's the string, okay, and so what that's going to do is behind the scenes it's going to create a column, that's actually a number right as an integer and it's going to store a mapping from the integers to the streets. Okay, the reason it's trained cats, as it uses for the training set more advanced usage, is that when we get to looking at the test and validation sets, this is really important idea. In fact, Terrence came to me the other day and he said my models not working. Why not? And he figured it out for himself - it turned out the reason. Why was because the mappings he was using from string to number in the training set were different to the mappings? He was using from string to number in the test set. So therefore, in the training set, hi might have been three, but in the trait test set. It might have been two, so the two were totally different, and so the model was basically non predictive. Okay, so I have another function, called apply categories where you can pass in your existing training set and it all use the same mappings to let you all make sure your test set of validation set uses the same mappings okay.

So when I go trained cats, it's actually not going to make the data frame look different at all, but behind the scenes it's going to turn them all into numbers. When we finish at 12 11:50 - let's see here, we go I'll, try to finish on time. So you'll see now remember I mentioned there was this dot DT attribute that gives you access to everything, assuming it's a date time about the date time. There's a dot count attribute that gives you access to things as something's, a category all right and so usage banned. Was a string and so now that I've run train cats, it's turned it into a category, so I can go to your or usage banned cat right and there's a whole bunch of other things. We've got there, okay, so one of the things we've got, there is dot categories and you can see here is the list. Now one of the things you might notice, it's that this list is in a bit of a weird order: high low medium. The truth is it doesn't matter too much, but what's going to happen when we use the random forest, is it's actually good that this is going to be 0? This is going to be 1. This is gon na, be true and we're going to be creating decision. Trees and so we're going to have a decision tree that can split things at a single point, so it either be high versus low and medium or medium versus high and low. That would be kind of weird right. It actually turns out not to work too badly, but it'll work a little bit better if you have these in sensible orders.

Okay. So if you want to reorder a category, then you can just go caps net categories and pass in the order you want until it is ordered, and almost every pandas method has an in-place parameter, which, rather than returning a new data frame, is going to change that Data frame, okay, so I'm not going to do that. Like I didn't check that carefully for categories it should be ordered, but this seems like a pretty obvious one. You reiterate that issue. I don't understand what the chart so um the usage banned column. It's actually going to be. This is actually what I random forest is gon na see these numbers one zero, two one: okay and they map to the position in this array and as we're going to learn shortly, a random forest consists of a bunch of trees. It's going to make a single split and the single split is going to be either greater than or less than 1 or greater than a less than two right, so we could split it into high versus low and medium, which that semantically makes sense it's like. Is it big or we could split it into medium versus high and low? It doesn't make much sense right, so in practice, the decision tree could then make a second split to say like medium versus high and low and then within the high and low into high and low, but by putting it in a sensible order if it wants to Spit out

low, it can do it in one decision rather than two and we'll be learning more about this shortly.

It's it honestly, it's not a big deal, but I just wanted to mention it's there and it's also good to know that people when they talk about like different types of categorical variable, specifically, you need to know, there's a kind of categorical variable called ordinal and an Ordinal categorical variable is one that has some kind of order like high medium and low. Okay and random. Forests are terribly sensitive for that fact, but it's worth noting it's there and trying it out still ordering wouldn't sell for maximum that. That's what I'm saying it helps a little bit right. It means you can get there with one decision rather than two. I noticed there is a negative one in that list of categories. Is that, like an NA yeah exactly so for free, we get a negative one which prefers to missing and what are the things we're going to do? Is we're going to actually add one? Can somebody pass the vector Paul? Is we're going to add one to our codes? Maybe in two guys let people know it's coming yeah, so let people know we're going to add one to all of our codes to make missing a zero later on yeah we're going to get to that yeah yeah. So get dummies which we'll get to in a moment, is going to create three separate columns, ones and zeros for high once there's a million ones in series for low. Where else this one creates a single column with an integer zero one or two we're going to get to that one shortly. Yep did you have a question to Paul or just pointing out? Okay, okay.

So at this point, as long as we always make sure we use dot cat codes, the thing with the numbers in we're basically done all of our streams have been turned into numbers. Dates been turned into a bunch of numeric columns, and everything else is already a number okay. The only other main thing we have to do is notice that we have lots of missing values. So here is DFA. Is null that's going to return true or false, depending on whether something is empty dot? Some is going to add up how many empty for each series and then I'm going to sort them and divide by the size of the data set. So here we have some things which have like quite high percentages of Nantz, sir, so missing values we call them in play or what I call it. Maybe I didn't run it there. We go okay, so we're going to get to that in a moment, but I will point something out which is reading the CSV talk a minute or so the processing took another ten seconds or so from time to time. When I've done a little bit of work. I don't want to wait for again. I will tend to save where I'm at so here, I'm going to save it and I got to save it in a format called feather format. This is very, very new all right, but what this is going to do is it's going to save it to disk in exactly the same basic format, but it's actually in RAM. This is by far the fastest way to save something in the fastest way to read it back right.

So most of the folks you deal with unless they're on the cutting edge won't be familiar with this format, so this would be something you can teach them about. It's becoming the standard right, it's actually becoming something that's going to be used, not just in pandas, but in Java, in spark in lots of like things for like communicating across computers, because it's incredibly fast and it's actually co-designed by the guy that made Panthers by where's Mckinney, so we can just go deer 4.2 feather and pass in some name. I tend to have a folder called temp for all of my like as I'm going along stuff, and so when you go OS, don't make do as you can pass in any path path. Here you like, it won't complain if it's already there exists, okay equals true. If there are some sub directories, it'll create them for you. So this is a super. Handy little function, okay, so it's not installed so because I'm using Crestle for the first time. It's complaining about that. So if you get a message that something's not installed, if you're using anaconda, you can condor, install Crestle actually doesn't use anaconda, it uses a pit you, and so we wait for that to go along okay, and so now, if I run it and so sometimes

12. 01:07:14

- **Pre-processing to replace categories with their numeric codes,**
- **Handle missing continuous values,**
- **And split the dependant variable into a separate variable.**
- **proc_df() and fix_missing()**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

You may find you actually have to restart Jupiter, so I won't do that now, it's really out of time. So, if you restart Jupiter you'll be able to keep moving a lot so from now on, you don't have to rerun all the stuff they love. You could just say: PD, don't read further and we've got our data frame back. So the last step we're going to do is to actually replace the strings with their numeric codes and we're going to pull out the dependent variable sale price into a separate variable and we're going to also handle missing continuous values. And so how are we going to do that so you'll see here, we've got a function called proc DF. What is that croc DF? So it's in so fastai, dot, structured again, and here it is so quite a lot of the functions have a few additional parameters that you can provide and we'll talk about them later, but basically we're providing the data frame to process and the name of the Dependent variable that the the Y field - name - okay, and so what it's going to do is it's going to make a copy of the data frame? It's going to grab the Y value, it's going to drop the dependent variable from the original, and then it's going to fix missing. So how do we fix missing? So what we do to fix missing is pretty simple. If it's numeric, then we fix it by basically saying, let's first of all check that it does have some missing right, so if it does have some missing values, so in other words the is not some is nonzero, then we're going to create a new column called With name as the original plus underscore na, and it's going to be a bullion column with a 1 anytime that was missing and a 0 anytime.

It wasn't we're going to talk about this again next week, but this is, you know, give you the quick version. Having done that, where they're going to replace the NA s, the missing with the median okay so anywhere that used to be missing will be replaced with the median or add a new column to tell us which ones were missing. We only do that for numeric. We don't need it for categories because pandas had is handles categorical variables automatically by setting them to minus one. So what we're going to do is if it's not numeric and it's a categorical type, we'll talk about the maximum number of categories later but lets us units is always true. So if it's not a numeric type, we're going to replace the column with its codes, the integers, okay, plus one right so the by default pandas uses minus one for missing. So now zero will be missing and one two three four will be all the other categories. So we're going to talk about dummies later on in the course, but basically optionally. You can say that if you already know about dummy values, there are columns with a small number of possible values you can put in two dummies. Instead, you can America lysing them, but we're not going to do that for now. Okay, so for now all we're doing is we're using the categorical codes plus one replacing missing values with the median, adding an additional column telling us which ones were replaced and removing the dependent variable. So that's what proc DF does runs very quickly.

Okay, so you'll see now sale price is no longer here. Okay, we've now got a whole new color, a whole new variable called Y, the contain sale press you'll see we put a couple of extra blah underscore na s at the end. Okay, and if I look at that, everything is a number okay. These boolean z' are treated as numbers. They're just considered contributed a zero, or one that is displayed as false and true they can see here is at the end of a month, is at the start of a

month, is at the end of a quarter. It's kind of funny right because we've got things like a model ID which presumably is something like that could be a serial number. It could be like the model identifier, that's created by the factory or something we've got like a data source ID like some of these are numbers but they're, not continuous. It turns out actually random forests work fine with those we'll talk about. Why and how and a lot about that in detail, but for now all you need to know is no problem. Okay, so as long as this is all numbers which it now is, we can now go ahead and create a random forest, so m dot random forest regressor random forests are trivially paralyse abour. So what that means is that they, if you've, got more than one CPU, which everybody will basically on their computers at home and if you've got a t2 dot medium or bigger at AWS. You've got multiple CPUs trivially paralyse Abul means that it will split up the data across your different CPUs and basically linearly scale right.

So the more CPUs you have pretty much. It will divide the time it takes by that number, not exactly but roughly so n jobs equals minus one tells the random forest regressor to create a separate job. It's a separate process, basically for each CPU. You have so that's pretty much what you want all the time fit the model using this new data frame we created using that Y value, we pulled out and then get the score. Ok, the score is going to be the r-squared we'll define that next week. Hopefully, some of you already know about the r-squared. One is very good. Zero is very bad, so, as you can see, we've mmediately got a very high score. Okay, so that looks great. But what we'll talk about next week, a lot more, is that it's not quite great, because maybe we had data that had points that look like this and we fitted a line that looks like this when actually we want to want normal, it looks like that. Ok, the only way to know whether we've actually done a good job is by having some other data set, that we didn't use to train the model. Now we're going to learn about some ways with random fire, we can kind of get away without even having that. Other data set, but for now what we're going to do is we're going to

13. 01:14:01

- **'split_vals()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Split into twelve thousand rows, which we're going to put in a separate data set called the validation set versus the training sets going to take, contain everything else right, and our data set is going to be sorted by date, and so that means that the most recent Twelve thousand rows are going to be our validation set again. We'll talk more about this next week. It's a really important idea, but for now we can just recognize that if we do that and run it, I've created a little thing called print score and it's going to print out the root mean square error between the predictions and actuals for the training set for The validation set that r-squared for the training set and the validation set and you'll see that actually the r-squared for the training was 0.98, but for the validation was 0.89 okay, then the RMS see and remember this is on the logs was point: oh nine, for the Training set 0.25 for the validation set. Now, if you actually go to cattle and go to the leaderboard okay, let's do it right now. He's got private and public. I click on public leaderboard and we can go down and find out. Where is point two five, so there are four hundred seventy-five teams and, generally speaking, if you're in the top half of a capital, competition you're doing pretty well so a point two-five here we are point two five: what was it exactly point? Two five by two 507 yeah about a hundred and tenth so we're about in the

top 25 %. So so the idea like this is pretty cool right with with like, with no thinking at all, using the defaults of everything we're in the top 25 % of a caracal competition, so like random, forests are insanely powerful, and this totally standardized process is insanely good.

For like any datasets, so we're gon na wrap up well, what I'm going to ask you to do for Tuesday, it's like take as many tackle competitions as you can, whether they be running now or old ones or datasets. That you're interested in for your hobbies will work and - and please try it right, try this process and if it doesn't work, you know tell us on the forum: here's the data, so I'm using here's where I got it from his like the stack trace of where I got an error or here's like you know, if you use my print score function or something like it like you know, show us what the training versus tests it looks like we'll, try and figure it out right, but what I'm hoping we'll find is that all Of you will be pleasantly surprised that, with with the you know hour or two with information you got today, you can already get better models than most of the very serious practicing data scientists that competing table competitions. Okay, great good luck and I'll see you on the forums. Oh one more thing, Friday. The other class said a lot of them had class during my office hours. So if I made them one till three instead of two two or four on Fridays, is that okay seminar, oh okay, I have to find a whole another time all right. I will talk to somebody who actually knows what they're doing, unlike me, about finding other cells.

Absolutely

Lesson 02

Outline

- Python basics
- Git, Symlink, AWS
- Python notebook basics
- Crash course on pandas
- FastAI introduction
- add_datepart
- train_cats
- Feather Format
- Run your first Random Forest

Video Timelines and Transcript

1. 00:03:30

- **simlink sim link to fastai directory**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So from here the next two or three lessons we're going to be really diving deep into random forests. So so far all we've learned is there's a thing called random forests for some particular datasets. They seem to work really really well without too much trouble, but we don't really know yet like well. How do they actually work? What do we do if they don't work properly? What are their pros and cons? What are the can we tune and so forth? So we're gon na look at all that and then after that, we're going to look at how do we interpret the results of random forests to get not just predictions but to actually deeply understand our data in a model driven way. So that's where we're going to go from here, so, let's just review where we're up to so we learned that there's this library called fastai and the fastai library is basically it's a highly opinionated library, which is to say, we've spent a lot of time. Researching what are the best techniques to get like state-of-the-art results, and then we take those techniques and package them into pieces of code so that you can use the state-of-the-art results yourself, and so, where possible, we wrap or provide things on top of existing code, and so In particular for the kind of structured data analysis, we're doing, scikit-learn has a lot of really great code, so most of the stuff that we're showing you from fastai is stuff to help us get stuff into scikit-learn and then interpret stuff out from scikit-learn. The fastai library, the way it works in our environment here, is that we've got out. Our notebooks are inside fastai, repo / courses and in / ml 1, + dl, 1 and then inside there there's a symlink to the parent of the parent, fastai. So this is a sim link to a directory containing a bunch of modules. So if you want to use the fastai library in your own code, there's a number of things you can do, one is to put your notebooks or scripts in the same directory as ml 1 or 0 1, whether it's already this simile and just import it Just like I do, you could copy this directory dot, dot, slash, dot, dot, slash birthday I into

somewhere else and use it or you could sim link it just like I have from here to wherever you want to use it so notice, it's mildly, confusing.

There's a github repo called fastai and inside the github repo caught fastai, which looks like this. There is a folder called fastai, okay and so the FASTA, a folder in the FASTA, a repo contains the FASTA, a library and it's that library. When we go from fastai dot imports to import star, then that's looking inside the farsi, a folder for a file called inputs, imports, py and importing everything from that. Okay, yes, Danielle, okay, and just like as a clarifying question for this Bentley. It's just that anything! That's just the Ellen thing that you talked about yeah, so a symlink is something you can create by typing: Ln minus s and then the path to the source, which in this case would be dot, dot, dot. First, AI could be relative or it could be absolute and then the name of the destination. If you just put the current directory at the destination, it'll use the same name as it comes from like a alias on the Mac or a shortcut on Windows, and when you do the important system yep got it imports this and then append that relative link. That also creates the same link and I don't think I've created the same link anywhere in the workbooks. The symlink actually lives inside the github repo okay. I created some symlinks in the deep learning notebook to some data that was different at the top of Tim Lee's workbook from the last class. There was important, oh yeah, don't do that? Probably I mean you you can, but I think this is.

I think this is better like this way you can go from fastai imports and regardless of kind of how you got it there, it's it's going to work. You know okay, okay, so then we had all of our data for blue books for bulldozers, competition in data, slash bulldozers, and here it is right, so we were able to read that CSV file. The only thing we really had to do was to say which columns were dates and having done that, we were able to take a look at a few earth examples of the rows of the data, and so we also noted that it's very important to deeply understand The evaluation metric for this project, and so if a cattle, they tell you what the evaluation metric is, and in this case it

2. 00:06:15

- **understand the RMSLE relation to RMSE, and why use np.log('SalePrice') with RMSE as a result**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Was the root mean squared log error, so that is some of the actuals the predictions now, but it's the log of the actuals, the log, the predictions squared right. So if we replace actuals with log actuals and replace predictions with log predictions, then it's just the same as root mean squared error. So that's what we did was we replaced sale price with log of sale, price and so now, if, if we optimize for root, mean squared error, we're actually optimizing for the root mean squared error of the logs okay. So then we learnt that we need all of our columns to be numbers, and so the first way we did, that was to take the date column and remove it and instead replace it with a whole bunch of different columns such as is that date. The start of a quarter is at the end of a year. How many days are elapsed since generally, the first 1970, what's a year, what's the month or the day of weeks and so forth? Okay, so they're all numbers, then we learnt that we can use train underscore katz to replace all of the strings with categories. Now, when you do that, it doesn't look like you've done anything different. They still look like strings all right, but if you actually take a deeper look, you'll see that the

datatype now is not string, but category and category is a pandas class where you can then go dark cat dot and find a whole bunch of different attributes such As cat categories to find a list of all of the possible categories - and this says hi is going to be 0 low - will become 1. Medium will become 2, so we can then get codes to actually get the numbers.

So then, what we need to do to actually use this data set to turn it into numbers, is take every categorical column and replace it with cap codes, and so we did that using

3. 00:09:01

- **proc_df, numericalize**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Proc DF, okay. So how do I get the source code for proc DF question: question: mark? Okay, all right! So if I scroll down, I go through each column and I numerical eyes it. Okay, that's actually the one I want. So I'm going to now have to look up numerical eyes so tab to complete it. If it's not numeric, then replace the data frames filled with that columns cat codes last one because otherwise unknown is minus one. We went unknown to be zero. Okay, so that's how we turn the strings into numbers right. They get replaced with a unique, basically arbitrary index. It's actually based on the alphabetical order of the feature names. The other thing property f did remember, was continuous columns that had missing values. The missing got replaced with the median and we added an additional column called column name underscore na, which is a boolean column, told you if that particular item was missing or not so once we did that we were able to call random forest regressor dot fit and Get the dots poor and turns out, we have an r-squared of 0.98. Can anybody tell me what an r-squared is? Listen um? Can we

4. 00:11:01

- **rsquare root square of mean errors RMSE,**
- **What the formula rsquare (and others in general) does and understand it**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

You so r-squared essentially shows how much variance is explained by the modal. This is the yeah. This is the relation of this is ssro, which is like trying to trying to remember the exact formal, but I don't roughly a curative way, yeah intuitively. It's how much the model explains the how much it accounts for the variance in the data. Okay, good. So let's talk about the formula, and so with formulas the idea is not to learn the formula and remember it, but to learn what the formula does and understand it right. So here's the formula: it's 1 minus something divided by something else. So, what's this something else from the bottom, SS taught okay. So what this is saying is we've got some actual data, so my eyes right, we've got some actual data, 3e two-for-one, okay and then we've got some average okay. So our top bit, this SS tot is the sum of each of these that so, in other words, it's telling us. How much does this data bury? Perhaps more interestingly, is I remember when we talked about like last week. What's the simplest non stupid model you could come up with and I think the simplest non stupid model we came up with was create a column of the mean just copy the mean a

bunch of times and submit that to cable. If you did that, then your root mean squared error would be this, so this is the root mean squared error of the most naive non stupid model, where the model is just predict mean on the top.

We have SS res, which is here, which is that we're now going to add a column of predictions, okay, and so then what we do is, rather than taking the. Why I, why mean we're going to take? Why I Fi right - and so now, instead of saying, what's the root mean squared error of our naive model, we're saying: what's the root mean squared error of the actual model that we're interested in and then we take the ratio, so in other words, if we actually Were exactly as effective as just predicting the mean, then this would be top and bottom would be the same. That would be 1. 1 minus 1 will be 0. If we were perfect so fi minus y, I was always zero. Then that's zero, divided by something one. Minus that is 1 ok, so what is the possible range of values of R squared? Okay? I heard a lot of 0 to 1. Does anybody want to give me an alternative negative 1 to 1 anything less than one there's the right answer, let's find out way through the boss? Okay, so why is it any number less than one which you can make a model? Basically, as crappy as you want, and just I guess like the arrows as you want you're just subtracting from one in the formula exactly so. Interestingly, I was talking to our computer science, professor Terrence this morning, who was talking to where a statistics professor told him that the possible range of values was asked where it was zero to one. I said that is totally not true.

If you predict infinity for every column, sorry for every row, then you're going to have infinity for every residue and so you're going to have one minus infinity. Okay, so the possible range of values is less than one. That's all we know, and this will happen you will get negative values, sometimes in your r-squared and when that happens, it's not a mistake. It's not it's not like a bug. It means your moral is worse than predicting the main okay, which is suggest it's not great. So that's R, squared it's not it's not necessarily what you're actually trying to optimize right, but it's it's it's the nice thing about it is that it's a number that you can use kind of for every model, and so you can kind of start try to get A feel of like what does point-eight look like what does point-9 look like so like something I find interesting is to like create some different. Synthetic data sets just to two dimensions, with kind of different amounts of random noise and like see what they look like on. A scatterplot and see what they are squared are just gon na get a feel for like what does it ask for it? You know, is it a spur to point line close or not about 0.7 closed or not? Okay, so I think r-squared is a useful number to have a familiarity with, and you don't need to remember the formula if you remember the meaning, which is: what's the ratio between how good your model is, it means bit error versus how good is the naive mean Model for it,

5. 00:17:30

- **Creating a good validation set, ‘split_vals()’ explained**
- **“I don’t trust ML, we tried it, it looked great, we put it in production, it didn’t work” because the validation set was not representative !**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Squared error, okay and LK is 0.98. It's saying it's a very good model. However, it might be a very good model because it looks like this all right. This would be called overfitting, so we may well have created a model which is very good at running through the points that we gave

it. But it's not going to be very good at running through points that we didn't give it. So that's why we always want to have a validation set. Creating your validation set is the most important thing that I think you need to do when you're doing a machine learning project, at least in terms of in the actual modeling, because what you need to do is come up with a data set where the scroller of Your model on that data set is going to be representative of how well your model is going to do in the real world, like in cattle on the leaderboard or off Kaggle like when you actually use it in production. I very very very often hear people in industries they, I don't trust machine loading. I tried modeling once it looked great, we put it in production, it didn't work now, whose fault is that right? That means their validation set was not representative right. So here's a very simple thing, which, generally speaking, cattle, is pretty good about doing. If your data has a time piece in it right, as happens in Blue Book for bulldozers in Blue Book for bulldozers, we're talking about the sale price of a piece of industrial equipment on a particular date.

So the start up during this competition wanted to create a model that wouldn't predict last February's prices. That would predict next month's prices, so what they did was they gave us data representing a particular date range in the training set, and then the test set represented a future set of dates that wasn't represented in the training set right. So that's pretty good right. That means that, if we're doing well on this model, we've built something which can actually predict the future, or at least it could predict the future. Then assuming things haven't changed dramatically. So that's the test set. We have so we need to create a validation set. That has the same properties, so the test set had 12,000 rows in so let's create a validation set that has 12,000 rows right and then, let's split the data set into the first n minus 12 thousand rows for the training set and the last 12,000 rows. For the validation set, and so we've now got something which hopefully looks like cackles test set, plus enough that when we actually try and use this validation set we're going to get some reasonably accurate scores and the reason we want this is because on cattle, you can Only submit so many times and if you submit too often you'll end up fitting to the leaderboard anyway and in real life you actually want to build a model. That's going to work in.

Why did you have a question? Can we help the green box? Can you explain the difference between a validation set and

6. 00:21:01

- **overfitting over-fitting underfitting ‘don’t look at test set !’,**
- **Example of failed methodology in sociology, psychology,**
- **Hyperparameters,**
- **Using PEP8 (or not) for ML prototyping models**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

A test set absolutely so. What we're going to learn today is how to set what other things alone. It's how to set high parameters. Hyper parameters are like tuning parameters that are going to change how your model behaves now, if you just have one holdout set, so one set of data that you're not using to train with, and we use that to decide which set of hyper parameters to use. If we try a thousand different sets of hyper parameters, we may end up overfitting to that holdout set, that is to say, well, find something which only accidentally worked. So what we actually want to do is we want to have a second holdout set where we can say: okay, I'm

finished okay. I've done the best I can and now just once right. At the end, I'm gon na see whether it works, and so this is something which almost nobody in industry does correctly. You really actually need to remove that holdout set. That's called the test set, remove it from the data, give it to somebody else and tell them do not. Let me look at this data until I promise you I'm finished like it's so hard, otherwise, not to look at it and, for example, in the world of psychology and sociology you might have heard about this replication crisis. This is basically because people in these fields have accidentally or intentionally maybe been peed hacking, which means they've been basically trying lots of different variations until they find something that works, and then it turns out and they try to replicate it.

In other words, it's like somebody creates a test set. Somebody says okay, this study, which shows you know the impact of whether you eat marshmallows, on your tenacity later in life, I'm going to rerun it and like over half the time they're finding the effect turns out. Not to exist, so that's why we want to have a test set, get that next door yeah. So for handling categorical data. You can market those to numerix two numbers order. Numbers I've seen a lot of models where we convert categorical data into different columns using one for encoding, yes, so which approach to use in which model yeah we're kind of track. For that today, yeah. It's a great question: okay, so so I'm splitting my my data into validation and training sets, and so you can see now that my validation set is twelve thousand about 66. Where else my training set is three hundred ninety nine thousand sixty-six okay. So we're going to use this set of data to train a model and this set of data to see how well it's working. So when we then tried that last week we found out just a moment. We found out that our model, which had point nine eight two R squared on the training, set only had point eight, eight, seven on the validation set, which makes us think that we're overfitting quite badly, but it turned out it wasn't too badly because the root mean Squared error on the logs of the prices actually would have caught us in the top 25 percent of the competition anyway.

So, even although we're overfitting, it wasn't the end of the world, could you pass the microphone to Marsha? Please? Ah, in terms of you dividing the set into training and validation, it seems, like you, simply take the first and train observations of the data status and set them aside. Why don't you like? Why don't you randomly pick up the observations? Because if I did that, I wouldn't be replicating the test set, so cattle has a test set that when you actually look at the dates in the test set, they are a set of dates that are more recent than any date in the training set. So if we used a validation set, that was a random sample. That is much easier because we're predicting options like what's the value of this piece of industrial equipment. On this day, when we add, we already have some observations from that day, so in general, anytime, you're building a model that has a time element. You want your test set to be a separate time period and therefore you really need your validation set to be observer time period as well, and in this case the data was already sorted. That's why this works. So let's say we have our test. The training set by which we in the data and then we have the validation set against which we are trying to find the r-square in in case of r-squared, turns out to be really bad. We would want to cheer to not parameters and run it again. Yes, so wouldn't that be eventually overfitting on the overall training set yeah.

So, actually, that's that's the issue, so that would eventually have the possibility of overfitting on the validation set and then, when you try it on the test set or we submit it to kaggle, it turns out not to be very good, and this happens in careful competitions. All the time careful actually has a fourth data set, which is called the private leader board set and every time you submit to Kaggle, you actually only get feedback on how well it does on something called the public leader board set, and you don't know which rows They are, and at the end of the competition

you actually get judged on a different data set entirely called the private leader board set. So the only way to avoid this is to actually be a good machine learning practitioner and know how to set these parameters as effectively as possible, which we're going to be doing gladly today and over the next few weeks. Can you get that actually watching? Is it too early or late to ask? What's the difference between a paper parameter and a parameter? Okay? So, let's start tracking things on written in spread era. So here is root, mean square error in a line of code, and you can see here like this - is one of these examples where I'm not writing this. The way a proper software engineer would write this right. So a proper software engineer would be a number of things differently. They would have it on a different line.

They would use longer variable names, they would have documentation blah blah blah right, but I really think like for me. I really think that being able to look at something in one go with your eyes and like over time, learn to immediately see what's going on, has a lot of value and also to like consistently use, like particular letters, to have many particular things or abbreviations. I think works really well in data science, if you're doing it like a take-home interview test or something you should write your code according to pet eight standards right. So pep eight is the the style guide for python code and you should know it and use it, because a lot of software engineers are super anal about this kind of thing, but for your own work, you know, I think this is. I think this works well for me, you know, so I just wanted to make you aware a that. You shouldn't necessarily use this as a role model for dealing with software engineers, but B that I actually think this is not. This is a reasonable approach. Okay, so there's a root mean

7. 00:29:01

- **RMSE function and RandomForestRegressor,**
- **Speeding things up with a smaller dataset (subset =),**
- **Use of '_' underscore in Python**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Squared error and then from time to time we're just going to print out the score which will give us the RMS C of the predictions on the training versus the actual. There are predictions on the valid versus the actual RMS, see the R squared for the training and the R squared of the Bell and we'll come back to over in a moment. So when we ran that we found that this rmse was in the top 25 %, and it's like okay there's a good start now. This took eight seconds of wall time, so eight actual seconds, if you put percent time it'll, tell you how long things talk, and luckily I've got quite a few calls quite a few CPUs in this computer because it actually took over a minute a compute time. So I paralyzed that across cause, if your dataset was bigger or you had less cause, you know you could well find that this took a few minutes to run or even a few hours, and my rule of thumb is that if something takes more 10 seconds to Run it's too long for me to do like interactive analysis with it right. I want to be able to like run something wait a moment and then continue. So what we do is we try to make sure that things can run in a reasonable time and then, when we're when we're finished at the end of the day, we can then say: ok, this feature engineering, these hyper parameters, whatever these are all working well and I'll now rerun it, you know, that's the big, slow, precise way.

So one way to speed things up is to pass in the subset parameter to proc DF and that will randomly sample my data right, and so here I'm got a randomly sample 30,000 rows. Now,

when I do that, I still need to be careful to make sure that my validation set doesn't change and that my training set doesn't overlap with the dates, otherwise I'm cheating. So I call split bells again to again do this split by dates and you'll also see I'm using rather than putting it into a validation set, I'm putting it into a variable called underscore. This is kind of a standard approach in Python is to use a variable called underscore. If you want to throw something away, because I don't want to change my validation set like no matter what different models I build, I want to be able to compare them all to each other, so I want to keep my validation set. The same all the time. Okay, so all I'm doing here is I'm resampling. My training set into a 20, the first 20,000 out of a 30,000 subsets, so I now could run that and it runs it 621 milliseconds. So I can like really zip through things now. Try things out. Okay, so with that, let's use this subset to build a model that is so simple that we can actually take a look at it, and so

8. 00:32:01

- **Single Tree model and visualize it,**
- **max_depth=3,**
- **bootstrap=False**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

We're going to build a forest is made of trees, and so before we look at the forest, we look at the trees in scikit-learn. They don't call them trees, they call them estimators. So we're going to pass in the parameter number of estimators equals one. The Creator forest with just one tree and then we're going to make a small tree, so we pass in maximum depth, equals three and a random forest as we're going to learn randomizes a whole bunch of things. We want to turn that off so to turn that off you say, bootstrap equals false. So if I pass you in these parameters that creates a small deterministic tree okay, so if I fit it and say print score, my a squared has gone down from point. Eight. Five, two point four: so this is not a good model, it's better than the mean model. This is better than zero right, it's not a good model, but it's a model that we can draw all right. So, let's learn about what it's built. So a tree consists of a sequence of binary decisions of binary splits, so at first of all decided to split on coupla system greater than or less than point five. That's boolean variable, so it's actually true or false, and then within the group we're a couple system was true: we decided to split into year, made greater than or less than 1987 and then we're a couple system was true, and year made was less than or equal To 1986 it used fi product class desk is less than or equal to 0.75 and so forth. Right so right at the top.

We have 20,000 samples, 20,000 rows right, and the reason for that is because that's what we asked for here when we split our data in the sample. I just want to double-check that for your decision tree, that you had there that the coloration was whether it's true or false, not so like it, it gets darker, it's true for the next one, not the darker is a higher value, we'll get to that. In a moment, okay, so let's look at these numbers here. So in the whole data set well our sample that we're using there are 20,000 rows. The meter, the average of the log of Christ is 10.1, and if we built a model where we just used that average all the time, then the mean squared error would be 0.477 okay, so this is, in other words, the denominator of an R squared all right. This is like the most basic model is a tree with zero splits right, which is just predict the average. So the best single binary split we can make it turns out to be splitting by where the coupler system is greater than or equal to sorry less than or equal to or greater than 0.5 know,

whether it's true or false and turns out. If we do that the mean squared error of couple system is less than 0.5, so it's false goes down from 0.477 to 0.1 one right. So it's really improved the error a lot in the other group. It's only improved at a bit, so I'm from 0.47 2.41, and so we can see that the coupler system equals false group has a pretty small percentage. It's only got twenty two hundred of the twenty thousand all right. Where else this other group has a much larger percentage, but it hasn't improved it as much so let's say you wanted to create a tree with just one split, so you're just trying to find like what is the very best single binary decision you can make for Your data, how might you be able to do that? How could you do it? I'm gon na give it to foot, specify the max depth of one, but I mean your writing.

You don't have a random first write. How are you going to? How are you going to like write? What's an algorithm, a simple algorithm which you could use sure, so we want to start building a random forest from scratch, so the first step is to create a tree. The first step to create a tree is to create the first binary decision. How are you gon na? Do it I'm gon na give it to Chris, maybe in two steps. So isn't this simply trying to find the best predictor based on maybe a linear regression? You could use a linear regression, but could you do something much simpler and more complete we're trying not to use any statistical assumptions here? I can't see your name, sorry and, of course, your friends anything can we just do like take just one variable if it is true give it like the true thing and if it is false so which variable are we gon na choose so at each binary point? We have to choose a variable and something to split on. How are we going to do that? Then I pass it over there. How do I pronounce your name chicken, so the variability tools could be like which divides population into two groups, which kind of heterogeneous to each other and homogeneous within themselves like having the same quality within themselves, and they are very different. Could you be more specific, like in terms of the target variable, maybe yeah right? Let's say we have two groups of tested, so one has a different price altogether from the second group. Yes, internally, they have similar prices.

Okay, that's good, so like to simplify things. A little bit when we're saying find a variable that we could split into such that the two groups are as different to each other as possible and okay, how do you? How would you pick which variable and which split point? That's the question: yeah. What's your first cut, which variable and which split point we don't like we're making the trade from scratch. We want to create our own tree. That makes sense. We've got somebody amazing. Can we test all of the possible split and see which one has this? Ah, small and rmse that sounds good okay. So let's dig into this so when you say test all of the possible splits, what does that mean? How do we enumerate all the possible splits, oh and think of that? But if you want for each variable, you could put one aside and then put a second aside and compare the two and if it was better good okay, so for each variable for each possible value of that variable see whether it's better now coming back to Maslak. So I want to dig into the better when you said see if the RMS see is better. What does that mean, though, because after a split you've got two rmse, so you've got two two groups, so you just gon na fit what that one variable comparing to the other, it's not so so. What I mean here is that before we decided to speed on coupler system, we had a root, mean square root of 0.477 and after we've got two groups, one were the mean square error point one another with a mean squared error of 0.4, so you treat each Individual model separately so for the first player you're, just gon na compare between each variable in himself and then you move on to the next node with the remaining, but but even the first node like so.

The model with zero splits has a single root, mean squared error, the model with one split. So the very first thing we tried. We've now got two groups with two mean square errors. You

don't give it to Daniel. Do you pick the one that gets them as different as they can be? Well we're truck? Well, okay, that would be one idea, get the two mean squared errors as different as possible, but why might that not work? What might be a problem with that sample size? Come on because you could just literally leave one point out yeah, so we could have like year made is less than 1950 and it might have a single sample with a low price and like that's, not a great split. Is it? You know because the other group is actually not going to be very interesting at all. Can you improve it? A bit. Can Jason improve it a bit? Could you take a weighted average yeah, a weighted average, so we could take point four one times: 17,000 plus 0.1 times 2,000. That's good right, and that would be the same as actually saying I've got a model. The model is a single binary decision and I'm going to say for everybody with year made less than ninety six point: five, I'm going to fill in point ten point two for everybody else are going to fill in nine point two and then I'm going to calculate The root mean squared error of this crappy model, and that would give exactly the same right as the weighted average that you're suggesting okay good.

So we now have a single number that represents how good a split is, which is the weighted average of the mean squared errors of the two groups that creates okay, and thanks to, I think it was. Was it Jake? We have a way to find the best split, which is to try every variable and to try every possible value of that variable and see which variable and which value gives us a split with the best score that makes sense. Okay, what's your name, sir okay and somebody give Natalie the box when you see every possible number for every possible variable link? Are you saying like here? We have 0.5 as like our criteria, to split the tree. So are you? Are you saying we're trying out a reasonable number for every possible value right? So cupola system only has two values, true and false. So there's only one way of splitting, which is trues and falses ear made, is an integer which varies between like O'donnell, 1960 and 2010. So we can just say what are all the possible unique values of year made and and try them all so we're trying all the possible spec points. Can you pass a better annual pass to me and I so I just want to clarify again for the first split. Why did we split on Cutler system, true or false sister, because what we did was we used Jake's technique? We tried every variable for every variable. We tried every possible split, so each one we noted down.

I think it was Jason's idea, which was the weighted average mean squared error of the two groups are created, we found which one had the best means grid error and we picked it and it turned out. It was cutlass system. True or false. Does that make sense? I guess my question is more like so coupler system is like one of them like best indicators. I guess it's the best. We tried every variable and every possible level everything else at try. It wasn't as good okay and then you do that right. So now that we've done that, we now take this group here, everybody who's got coupler system equals true, and we do it again for every possible variable for every possible level for people where coupler system equals true, what's the best possible split and then are there circumstances When it's not just like binaries, like you split it into like three groups for like example, you're made, so I'm gon na make a claim, and then I'm gon na see if you can justify it, I'm gon na claim that it's never necessary to do more than One split at a level because you can just cuz: you can just split them again exactly so, you can get exactly the same result by submitting twice okay good. So that is the entirety of creating a decision tree. You stop either when you hit some limit. That was requested, so we had a limit where we said max depth equals three, so that's one one way to stop would be you ask to stop at some point, and so we stopped otherwise you stop when you're your leaf nodes.

These things at the end of court leaf nodes when your leaf nodes only have one thing in them: okay, that's a decision tree. That is how we grow a decision tree, and this decision tree is not

very good because it's got a validation r-squared of 0.4. So we could try to make it better by removing next steps, equals three right and creating a deeper tree. So it's going to go all the way down they're going to keep splitting these things further until every leaf. Node only has one thing in it and if we do that, the training r-squared is, of course one because we can exactly predict every training element, because it's in a leaf node or on its own, but the validation r-squared is not one. It's actually better than our really really shallow tree, but it's not as good as we'd like okay. So we want to find some other way.

9. 00:47:01

- **Bagging of little Boostraps, ensembling**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Of making these trees better and the way we're going to do it is to create a forest. So what's the forest to create a forest, we're going to use a statistical technique called bagging and you can bag any kind of model. In fact, Michael Jordan, who is one of the speakers at the recent recent data Institute conference here at university of san francisco, developed a technique called the bag of little bootstraps and which he shows how to use bagging for absolutely any kind of model to make it More robust, and also to give you confidence intervals, the random forest is simply a way of bagging trees. So what is bagging managing is a really interesting idea, which is what, if we created five different models, each of which was only somewhat predictive but the models weren't at all correlated with each other. They gave predictions that weren't correlated with each other. That would mean that the five models would have profound different insights into the relationships in the data, and so, if you took the average of those five models right, then you're effectively bringing in the insights from each of them. And so this idea of averaging models is a is: is a technique for ensemble right, which is really important? Now, let's come up with a more specific idea of how to do this. Ensemble. What if we created a whole lot of these trees, big, deep massively overfit trees right, but each one? Let's say we only pick a random one-tenth of the data.

So we pick one out of every 10 rows at random, build a deep tree right, which is perfect on that subset and kind of crappy. On the rest, all right, let's say we do that a hundred times so different random sample every time. So all of the trees are going to be better than nothing right because they do actually have a real random subset of the data, and so they found some insight but they're also overfitting terribly, but they're all using different random samples. So they all over fit in different ways on different things, so in other words, they all have errors, but the errors are random. What is the average of a bunch of random errors? Zero, so in other words, if we take the average of these trees, each of which have been trained on a different random subset, the errors will average out to zero and, what's left is the true relationship and that's the random forest right. So there's the technique right. We've got a whole bunch of rows of data. We grab a few at random, all right put them into a smaller data set and build a tree based on that. Okay - and then we put that tree aside and do it again with a different random subset and do it again with a different random subset. Do that a whole bunch of times and then for each one. We can then make predictions by running our test data through the tree to get to the leaf mode. Take the average in that leaf, node for all the trees and average them all together.

So, to do that, we simply call random forests regressor and by default it creates n what scikit-

learn cords estimators an estimator is a tree right, so this is going to create ten treats, and so we go ahead and train it. I can't remember if I remember, to split okay, so create our ten trees and we're just doing this on our little random subset of 20,000, and so, let's take a look at one example, can you pass the box to Devin so just to make sure I understand This so you're saying like we take ten kind of crappy models. We average ten crappy models and we get a good model exactly because the crappy models are based on different random subsets, and so their errors are not correlated with each other. If the error errors work, are they with other this isn't going to work okay? So the key insight here is to construct multiple models which are better than nothing and where the errors are as much as possible, not correlated with each other. So is there like a certain number of trees that, like we need that, in order to be this sort of the things like valid or invalid, there's like has a good validation set, our MSC or not, you know, and so that's what we're going to look at Is how to make that metric higher, and so this is the first of our hyper parameters, then we're going to learn about how to tune hyper parameters and the first one is going to be the number of trees and we're about to us. Look at that now.

Yes, mostly you're selecting, aren't they exclusive? Yes, so I mentioned you know. One approach would be pick out like attempt at random, but actually what scikit-learn does by default is for n rows. It picks out n rows with replacement, okay and that's called bootstrapping, and if memory serves me correctly, that gets you on average, 63.2 % of the rows will be represented, and you know a bunch of them will be represented multiple times yeah. It sure so, rather than just picking out like a tenth of the rows at random, instead we're going to pick out of an n row data set we're going to pick out n rows with replacement, which, on average, gets about 63. I think 63.2 % of the rows will be represented. Many of those rows will appear multiple times. I think there's a question behind you. In essence, what this model is doing is, if I understand Craig, is just picking out data points that look very similar to the one you're looking at yeah, that's a great insight. So what a tree is kind of doing, there's no quite complicated way of doing about doing that. I mean there would be other ways of assessing similarity. There are other ways of assessing similarity, but what's interesting about this way, is it's doing it in in tree space? Right so we're basically saying what are in this case like for this little tree.

What are the 593 samples you know closest to this one and what's their average closest in tree space, so other ways of doing that would be like and we'll learn later on. In this course about K, nearest neighbors, you could use like Euclidean distance C right, but here's a thing. The whole point of machine learning is to identify which variables actually matter the most and how do they relate to each other and to your dependent variable together right. So if you've like imagine a synthetic data set where you create five variables that add together to create your independent to create your dependent variable and ninety five variables, which are entirely random and don't impact your dependent variable. And then, if you do like a K, nearest neighbors in Euclidean space, you're going to get meaningless nearest neighbors, because most of your columns are actually meaningless or imagine your actual relationship is that your dependent variable equals x1 times x2. Then you'll actually need to find this interaction right, so you don't actually care about how close it is to x1 and how close to x2, but how close to the product. So the entire purpose of modeling in machine learning is to find a model which tells you which variables are important and how do they interact together to drive your dependent variable and so you'll find in practice the difference between using like tree space or random forest space? To find your nearest neighbors versus like Euclidean space is the difference between a model that makes good predictions in the model.

That makes many most predictions elicit here. I did, but I feel, like we've got only 35 minutes, so yeah great, so so in general, a machine learning model which is effective is one which is accurate. When you look at the training data, it's it's accurate at predicting at actually finding the relationships in that training data, and then it generalizes well to new data, and so in bagging. That means that each of your individual estimators at your individual trees, you want to be as predictive as possible, but for the predictions of your individual trees to be as uncorrelated as possible, and so the inventor of random forests talks about this at length. In his original paper that introduced this in the late 90s, this idea of trying to come up with predictive but poorly correlated trees. The the research community in recent years has generally found that the more important thing seems to be creating uncorrelated trees rather than more accurate trees. So more recent

10. 00:57:01

- **scikit-learn ExtraTreeRegressor randomly tries variables**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Advances tend to create trees which are less predictive on their own, but also less correlated with each other. So, for example, in scikit-learn there's another class you can use called extra trees regress on your extra trees, classifier with exactly the same API. You can try it tonight. Just replace my random prose regressor with that that's called an extremely randomized trees model and what that does is the same as what we just discussed, but rather than trying every speck of every variable. It randomly tries a few splits of a few variables right. So it's much faster, the Train, it has more randomness, okay, but then you've got time. You can build more trees and therefore get better generalization. So in practice, if you've got crappy individual models, you just need more trees to get a good end up model Melissa. Could you pass that over to Devin? Could you talk a little bit more about what you mean by like uncorrelated trees, yeah? If I build a thousand trees h1 on just ten data points, then it's quite likely that the ten data points for every tree are going to be totally different, and so it's quite likely that those ten trees are going to trees are going to give totally different Answers to each other, so the correlation between the predictions of tree, one and three two is going to be very small between tree one and three, three, very small and so forth. On the other hand, if I create a thousand trees where each time I use the entire data set with just one element removed, all those trees are going to be nearly identical.

Ie, their predictions will be highly correlated, and so, in the latter case, it's probably not going to generalize very well where else in the former case the individual trees we're not going to be very predictive. So I need to find some nice in-between, so yes, Danielle and is there a case where you want to use one over the other like any particular times, yeah so again, hyper parameter tuning so driven in terms of like random, random forests versus extremely randomized phase yeah. So again, a hyper parameter, walk tree architecture. Do we use so we're going to talk about that now? Can you pass that through? Do you know you know, I was just trying to understand how this random forest actually makes sense for continuous variables. I mean I'm assuming that you build a tree structure and the last final notes you'd be seeing like maybe the small node represents, maybe a category a or a category B. But how does it make sense for your continuous storage? So this is actually what we have here, and so the value here is the average. So this is the average log of price for this subgroup and that's what we do. The prediction is the average of the value of the dependent variable in that leaf.

Node means, finally, if you have just like dengue nodes, you just have 10 values. Yes, that's right! Well, if it was only one tree right, so a couple things to remember. The first is that by default, we're actually going to train the tree all the way down until the leaf nodes were size 1, which means for a data set with n rows.

We're gon na have n leaf nodes and then we're going to have multiple trees, which we average together right. So in practice, we're gon na have a you know: lots of different possible values. It's a question behind you! So far, the continuous variable. How do we decide like which value to split out? Because there could be many values? We try every possible value of that in the training set, one did be computationally computational expensive, and this is where it's very good to remember that your CPUs performance is measured in gigahertz, which is billions of clock cycles per second, and it has multiple cores and each Core has something called Simbi single instruction, multiple data, where it can direct up to eight computations per core at once, and then, if you do it on the GPU, the performance is measured in teraflops, so trillions of floating-point operations per second, and so this is where, when It comes to designing algorithms, it's very difficult for us mere humans to realize how stupid algorithms should be given how fast today's computers are so yeah, it's quite a few operations, but that trillions per second you hardly notice it Russia. I have a question so essentially at each remote we make a decision like which category T or which variable to use and which pinpoint yes, so we have MSE a calculated for each node.

So this is kind of our one of the decision criteria, but please MSE it is calculated for which model like which model underlies the model is the model is for the initial root mode is what if we just predicted the average right, which is here is ten Point, oh nine, eight and just the average, and then the next model is what, if we predicted the average of those people with capitalist system equals false and for those people with couple of system is true and then the next is what if we predicted the average Of couple systems equals true yeah made less than 1986. Is it always average sure we can use median or we can even run linear regression? There's all kinds of things we could do and practice the average works really well. There are. There are types of they're not called random forests, but there are kinds of trees where the leaf nodes are independent, linear, regressions, they're not terribly widely used, but there are certainly researchers who I have worked on them. Okay, thank you and pass it back over that afford and then to check. So this tree has a depth of three yeah and then I on one of the next commands we get rid of the max depth yeah the tree without the max depth. Does that contain the tree with with the depth of three yeah, so that is that, like by definition, it's yeah well, except in this case we've added randomness. But if you turned bootstrapping off then yeah the the deeper tree.

Will you know the the the the less deep tree would be, how it start and then it goes keep spitting okay. So you have many trees, you're gon na

11. 01:04:01

- **m.estimators_,**
- **Using list comprehension**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Have different leaf nodes across streets? Hopefully, so we want um. So how do you average

leaf nodes across different trees? So we just take the first row in the validation set. We run it through the first tree. We find its average nine point. Two eight then do it through the next tree find its average in the second tree, nine point: nine, five and so forth, and we're about to do that. So you'll see it okay. So let's try it right. So after you've built a random forest, each tree is stored in this attribute called estimators underscore okay, so one of the things that you guys need to be very, very comfortable with is using list comprehensions. Okay, so I hope you've all been practicing okay, so here I'm using a list comprehension to go through each tree in my model, I'm going to call predict on it with my validation set, and so that's going to give me a list of arrays of predictions. So H array will be all of the predictions for that tree and I have ten trees. Mp, dot stack, concatenates them together on a new axis. So after I run this and called shape, you can see. I now have the first axis ten means I have my ten different sets of predictions and for each one my validation set as a side of twelve thousand. So here are my twelve thousand predictions for each of the ten trees.

Alright. So let's take the first row of that and print it out, and so here are what weirdest thing here are ten predictions one from each tree: okay, and so then, if we say take the mean of that here is the mean of those ten predictions and then What was the actual? The actual was nine point. One. Our prediction was nine point O seven, so you see how like none of our individual trees had very good predictions, but the mean of them was actually pretty good, and so, when I talk about experimenting like Jupiter, notebook is great for experimenting. This is the kind of stuff I mean dig inside these objects and like look at them and plot them. Take your own averages cross check to make sure that they work the way you thought they did write your own implementation of r-squared make sure it's the same as a psychic learn version plot it like here's an interesting plot. I did let's go through each Taffet entries right and then take the mean of all of the predictions up to the ithe tree right. So, let's start by predicting just based on the first tree than the first two trees and the first three trees. And let's then plot the r-squared, so here's the r-squared of just the first tree is the r-squared of the first two trees, three trees, four trees, blah blah blah blah up to ten trees, and so not surprisingly, a script keeps improving right because the more estimators we Have the more bagging that we're doing the more it's? Well, it's going to generalize all right and you should find that that number there bit under point.

Eight six should match this number here: okay, let's rerun that yeah, okay, so they're actually slightly above what it says. All right so again, these are all like the cross checks. You can do the things you can visualize to deepen your understanding. Okay, so, as we add more trees, our r-squared improves, it seems to flatten out after a while. So we might guess that if we increase the number of estimators to twenty right, it's maybe not going to be that much better. So, let's see we've got point. Eight. Six. Two first is point: eight, six, oh yeah, so doubling the number of trees didn't help very much, but double it again. Eight six, seven double it again, eight six! Nine! So you can see like there's some point at which you're going to you know not want to add more trees, not because it's never going to get worse, because every tree is, you know, giving you more semi-random models to bag together right, but it's going to stop Improving things much okay, and so this is like the first hyper parameter. If you learn to set, is number of estimators and the method for setting is as many as you have time to fit, and that actually seemed to be hopping. Okay, now, in practice, we're going to learn to set a few more hyper parameters, adding more trees, slows it down, but with less trees. You can still get the same insights, so I've build most of my models in practice with like 20 to 30 trees, and it's only like then at the end of the project, or maybe at the end of the day's work.

I'll then try doing like. I don't know a thousand trees and run it overnight. Was there a

question? Yes, can we pass that Prince, so each tree might have different estimators different combination of estimators HQ is an estimator, so this is a synonym. So in scikit-learn, when I say estimator, they name three, so I mean each tree would have different break points on differently on different columns. But if, at the end, we want to locate the important features and we'll get to that yeah, so I'm after we finished with kind of setting hyper parameters. The next stage of the course will be learning about what it tells us about the data. If you need to know it now, you know, for your projects feel free to look ahead. There's a lesson to our F interpretation.

12. 01:10:00

- **Out-of-bag (OOB) score**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Them is where we can see it. Okay, so that's our first type of parameter um. I want to talk next about out-of-bag score. Sometimes your data set will be kind of small and you won't want to pull out a validation set because doing so means you now. Don't have enough data to build a good model. What do you do? There's a cool trick which is pretty much unique to random forests, and it's this. What we could do is recognize that some of our in our first tree some of our columns - sorry some of our rows - didn't get used. So what we could do would be to pass those rows through the first tree and treat it as a validation set and then for the second tree. We could pass through the rows that weren't used for the second tree through it to create a validation set for that and so effectively. We would have a different validation set for each tree and so now to calculate our prediction. We would average all of the trees where that row is not used for training right so for tree number one we would have the ones I've marked in blue here and then maybe for tree number. Two it turned out. It was like this one, this one, this one and this one and so forth right so as long as you've got enough trees, every rows going to appear in the out of bag sample from one of them, at least so you'll be averaging.

You know, hopefully, a few trees, so if you've got a hundred trees, it's very likely that all of the rows are going to appear many times in these other bag samples. So what you can do is you can create an out of bag prediction by averaging. All of the trees, you didn't use to train each individual row and then you can calculate your root, mean square error, R, squared, etc. On that, if you pass low B score equals true to scikit-learn, it will do that for you and it will create an attribute called oob score underscore, and so my little print score function here. If that attribute exists, it it adds it to the print. So if you take a look here, hoby square equals. True, we've now got one extra number and it's R squared that is the R squared for the oeob sample. It's a square. It is very similar, the R squared and the validation set, which is what we hoped for. Can we plus it? Is it the case that the prediction for the obese core has to be must be mathematically lower than the one for our entire forest um? Certainly, it's not true that the prediction is lower, it's possible for accuracy, yeah um. It's not mathematically necessary that it's true, but it's gon na be true on average, because your average for each row appears in less trees in the oeob samples and it does in the full set of trees. So, as you see here, it's a little less good.

So I'm in general, it's a great insight Chris in general, the OOBE r-squared will slightly underestimate how generalizable the model is the more trees. You add the less serious that

underestimation is and for me in practice. I I find it's totally good enough. You know in practice. Okay, so this

13. 01:13:45

- **Automate hyperparameters hyper-parameters with grid-search gridsearch**
- **Randomly subsample the dataset to reduce overfitting with 'set_rf_samples()', code detail at 1h18m25s**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Our B score is, is, is super handy and one of the things that super handy for is you're gon na see, there's quite a few hyper parameters that were going to set, and we would like to find some automated way to set them. And one way to do that is to do what's called a grid search. A grid search is where, if there's a scikit-learn function, called grid search and you pass in the list of all of the parameters, all of the hyper parameters that you want to tune. You pass in for each one a list of all of the values of that hyper parameter. You want to try and it runs your model on every possible combination of all of those hyper parameters and tells you which one is the best, and our B score is a great like choice for to forgetting it to Tory, which one is best in terms of Our V score, like that's an example of something you can do with a which works well now, if you think about it, I kind of did something pretty dumb earlier, which is. I took a subset of 30,000 rows of the data and it built all my models of that, which means every tree in my random forest is a different subset of that subset of 30,000. Why do that? Why not take a different? I, like a totally different subset of 30,000 each time, so in other words, let's leave the entire 300 thousand records, as is right, and if I want to make things faster right, pick a different subset of 30,000 each time so, rather than bootstrapping the entire set of Rows, that's just randomly sample a subset of the data, and so we can do that.

So let's go back and recall property, yet without the subset parameter to get all of our data again and so to remind you that is okay. 400,000. I, in the whole data frame of which we have three 89,000 in our training set, and instead we're going to go, set our F samples 20,000. Remember that was the site that off the 30,000 we use 20,000 of them in our training set. If I do this there now, when I run a random forest, it's not going to bootstrap an entire set of 391 thousand rows. It's going to just grab a subset of 20,000 rows, all right, and so now, if I run this, it will still run just as quickly as if I had like originally done a random sample of 20,000. But now every tree can have access to the whole data set right. So if I do enough estimators enough trees, eventually it's going to see everything right. So in this case, with 10 trees, which is the default. I get an R squared of 0.8 6, which is actually about the same as my R squared with the with the 20,000 subsets, and that's because I haven't used many estimators yet right. But if I increase the number of estimators, it's got to make more of a difference right. So if I increase the number of estimators two-forty, alright, it's going to take a little bit longer to run, but it's going

14. 01:17:20

- **Tip for Favorita Grocery competition,**
- **'set_rf_samples()',**
- **'reset_rf_samples()',**
- **'min_samples_leaf=',**

- **'max_features='**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

To be able to see a larger subset of the data set, and so, as you can see, the r-squared is gone up from point. Eight six two point: eight, seven, six: okay, so this is actually a great approach and for those of you who would doing the groceries competition, that's got something like 120 million roads, but there's no way you would want to create a random forest using 128 million roads. In every tree, like it's going to take forever, so what you could do is use this set area of samples to do like, I don't know a hundred thousand or a million I'll play around with it. So the trick here is that, with a random forest using this technique, no data set is too big. I don't care. If it's got a hundred billion rows right, you can create a bunch of trees, each one of the different random subsets. Can somebody pass the actual accuracy, so my question was for the albey scores and these ones does it take the only like for the ones from the sample or take from all the that's a great question, so unfortunately, scikit-learn does not support this functionality. Out of the box, so I had to write this and it's kind of a horrible hack right, because we're much rather be passing in, like a sample size parameter rather than doing this kind of setting up here. So what I actually do is, if you look at the source code, is I'm actually. This is a internal. This is the internal function.

I looked at their source code that they call and I've replaced it with a with a lambda function that has the behavior. We want, unfortunately, the current version is not changing how our B is calculated, so yeah, so currently low, B, scores and set our F samples are not compatible with each other, so you need to turn our B equals false. If you use this approach, which I hope to fix, but at this stage it's it's not fixed, so if you want to turn it off, you just call reset our F samples, okay, and that returns it back to what it was okay. So in practice, when I'm like doing interactive machine learning using random forests in order to like explore my model explore hyper parameters, the stuff we're going to learn in the future lesson where we actually analyze like feature importance and partial dependence and so forth. I generally use subsets and reasonably small forests, because all the insights that I'm going to get are exactly the same as the big ones, but I can run it in, like you know three or four seconds rather than hours alright. So this is one of the biggest tips I can give you and very very few people in industry or academia actually do this. Most people run all of their models on all of the data all of the time, using their best possible parameters, which is just pointless right if you're trying to find out like which features are important and how are they related to each other and so forth? Having that fourth decimal place of accuracy isn't going to change any of your insights at all, okay, so I would say, like do most of your models on you know a large enough sample size that your accuracy is.

You know reasonable when I say reasonable. It's like within a reasonable distance of the best accuracy you can get and it's taking you know a small number of seconds to train so that you can interactively. Do your analysis, so there's a couple more parameters. I wanted to talk about so I'm going to call reset RF samples to get back to our full data set because in this case at least on this computer, it's actually running in less than 10 seconds. So here's our baseline we're going to do a baseline with 40 estimators, okay, and so each of those 40 estimators is going to Train all the way down to all the leaf nodes. Just have one sample in them. So that's going to take a few seconds to run here we go so that gets us a point: eight nine, eight, a sweat on the validation set or 0.90 eight on the oeob. Now this case the OB is better. Why is it better? Well, that's because remember, our validation set is not a random sample. Our

validation set is a different time period. Okay, so it's actually much harder to predict a different time period than this one, which is just predicting random. Okay. So that's why this is not the way around we expected so the next. The first parameter we can try fiddling with is min samples leaf and so min samples leaf says, stop training the tree further. When your leaf node has three or less samples in there are going all the way down until there's one we're going to go down until there's three.

So in practice this means there's going to be like one or two less levels of decision being made, which means we've got like half the number of actual decision criteria we have to do. Is it's going to train more quickly? It means that when we look at an individual tree rather than just taking one point, we're taking the average of at least three points, that's who would expect the trees to generalize each one to generalize a little bit better okay, but each tree is probably going to Be slightly less powerful on its own, so let's try training that so possible values of min samples leaf. I find ones which work well are kind of 1. 3. 5. 10. 25. You know like I find that kind of range seems to work well but like sometimes, if you've got a really big data set and you're, not using the small samples. You know you might need a mint samples leaf of hundreds or thousands. So it's you kind of got to think about like how bigger your sub-samples going through and try things out now in this case, going from the default of one to three has increased. Our validation set our squared from 898 to 902. So it's a slight improvement. Okay and it's going to train a little faster as well as something else you can try, which is, and since this worked, I'm going to leave that in I'm going to add in max features equals point five. Why does max features do well? The idea is that the less correlated your trees are with each other, the better.

Now imagine you had one column that was so much better than all of the other columns of being predictive that every single tree you built, regardless of like which subset of rows, always started with that column. So the trees are all going to be pretty similar right. But you can imagine there might be some interaction of variables where that interaction is more important than that individual column. So if every tree always fits on the first thing, the same thing, the first time you're not going to get much variation in those trees. So what we do is, in addition to just taking a subset of rows. We then, at every single split point, take a different subset of columns. So it's slightly different to the row sampling for the row. Sampling H, nu tree is based on a random set of rows or columns sampling, every individual binary split. We choose from a different subset of columns, so in other words, rather than looking at every possible level of every possible column, we look at every possible level of a random subset of columns. Okay and each time each decision point each binary split. We use a different random subset. How many well you get to pick 0.5 means randomly choose half of them. The default is to use all of them. There's also a couple of special values you can use here.

As you can see in max features, you can also pass in square root to get square root of features or log two to get log two in features so in practice, good values. I found our range from 1.5 log to or square root. That's going to give you a nice bit of variation right. Can somebody positive Danielle and so just to clarify? Does that just like break it up smaller each time it goes through the tree or is it just taking half of what's left over or like hasn't been touched each time? There's no such thing as, what's left over after you've split on year, made less than or greater than 1984. You made still there right so later on. You might then spit on your maid left Center greater than 1989. So so it's just each time, rather than checking every variable to see where it's best fit. Is you just check half of them and so the next time you check a different hops. The next time you took a different path, but I mean, like terms is, as you got like further to like the leaf you're gon na have less options. No you're not you'd, never remove the

variables okay, you can use them again and again and again, because you've got lots of different spit points. So imagine, for example, that the relationship was just entirely linear between year made and price right, then, in practice to actually model that you know your real relationship is year made versus price right, but the best we could do would be this kind of. First of all, split here right and then just split here and here right and like spitting, spitting, split so yeah, even if the binary most random forest libraries don't do anything special about that they just kind of go. Okay, we'll try this variable! Oh it turns out.

There's only one level left, you know so yeah that didn't they don't do any kind of clever bookkeeping. Okay. Okay, so if we add next features equals 0.5, it goes up from 901, the 906, so that's better still, and so, as we've been doing, this we've also hopefully have noticed that our root mean squared error. Log price has been dropping on a validation set as well, and so it's now down to 0.2 to 86. So how good is that right? So, like our totally untrue and random florists got us in about the top 25 percent? Now remember, our validation set, isn't identical to the Kaggle test set right, and this competition unfortunately, is old enough that you can't even put in a kind of after this after the time entry to find out how you would have gone. So we can only approximate how we could have gone, but you know, generally speaking, is going to be a pretty good approximation. So 2:286 here is the competition. Here's, the public leaderboard two, two eight six here we go 14th or 15th place. So you know, roughly speaking, looks like we would be about in the top 20 of this competition with basically totally brainless random forest with some totally brainless minor, hyper parameter tuning, and so this is kind of why the random forest is such an important, not just first Step but often only step from the Xin learning, because it's kind of hard to screw it up like even when we didn't tune the hyper parameters, we still got a good result and then a small amount of type of parameter tuning got us a much better result And so any kind of model so, and I'm particularly thinking of like linear type models which have a whole bunch of statistical assumptions, and you have to get a whole bunch of things right before they start to work at all, can really throw you off track right Because they give you like totally wrong answers about how accurate the predictions can be, but also the random forest, you know, generally speaking, they tend to work on.

Most data sets most of the time with most sets of hyper parameters, so for

15. 01:30:20

- **Looking at 'fiProductClassDesc' column with `.cat.categories` and `.cat.codes`**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Example, we did this thing with it with our categorical variables. In fact, let's take a look at our tree. A single tree. Look at this right, fi product class desc less than 7.5. What does that mean? So f, I product plus desc, here's some examples of that column all right. So what does it mean to be less than or equal to 7? Well, we'd have to look at dark cat, dark categories to find out, okay, and so it's 0. 1. 2. 3. 4. 5. 6. 7. So what it's done is its created, a split where all of the backhoe loaders and these three types of hydraulics enter in one group and everything else is in the other group. So, like that's like weird, you know like like these aren't, even in order we could have made them in order if we had, you know, bother to say the categories have this order, but we hadn't right. So how come this even works like because when we turn it into codes, it's actually, this is actually what the random forest sees, and so imagine to think

about this. Imagine like the only thing that mattered was whether, as a hydraulic excavator at zero to two metric tons, and nothing else mattered. Imagine that right, so it has to pick out this this single level. Well, it can do that because, first of all, it could say: okay, let's pick out everything less than seven versus greater than seven to create.

You know this is one group, and this is another group right and then within this group they could then pick out everything less than six versus greater than six we're just going to pick out this one item right. So, with two split points: we can pull out a single category, so this is why it works. Right is because the tree is like infinitely flexible, even with a categorical variable, if there's particular categories which have different levels of price, it can like gradually zoom in on those groups by using multiple splits right now, you can help it by telling it the order of Your categorical variable, but even if you don't it's okay, it's just going to take a few more decisions to get there, and so you can see here it's actually using this product class desk quite a few times right and and as you go deeper down the tree. You'll see it used more and more right where else in a linear model or almost any kind of other model, certainly any any non tree model pretty much encoding. A categorical variable like this won't work at all, because there's no linear relationship to train totally arbitrary identifiers and anything right. So so these are the kinds of things that make Brennan forests very easy to use and and very resilient, and so by using that you know, we've gotten ourselves a model which is clearly you know world class.

At this point already, it's like you know, probably will, in the top 20 of this cackle competition and then in our next lesson, we're going to learn about how to analyze that model, to learn more about the data, to make it even better right. So this week try and like really experiment right, have a look inside, look, try and draw the trees, try and plot the different errors. Try maybe using different data, sets to see how they work really experiment, to try to get a sense and maybe try to like replicate things like write your own r-squared, you know, write your own versions or some of these functions. See if ya see how much you can really learn about your data set about the random person. Great see you on Thursday,

Lesson 03

Outline

- R^2 accuracy
- How to make validation sets
- Test vs. Validation Set
- Diving into RandomForests
- Examination of One tree
- What is 'bagging'
- What is OOB Out-of-Box score
- RF Hyperparameter 1: Trees
- RF Hyperparameter 2: max Samples per leaf
- RF Hyperparameter 3: max features

Video Timelines and Transcript

1. 00:02:44

- **When to use or not Random Forests (unstructured data like CV or Sound works better with DL),**
- **Collaborative filtering for Favorita**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Last lesson we looked at what random forests are, and we looked at some of the tweaks that we could use to make them work better. So, in order to actually practice this, we needed to have a Jupiter notebook environment running, so we can either install anaconda on our own computers. We can use AWS or we can use cress or comb that has everything up and running straight away, or else paper. Space comm also works really well, so assuming that you've got all that going, hopefully you had a chance to practice running some random forests. This week, I think one of the things to point out, though, is that before we did any tweets of any type of parameters or any tuning at all, the broad defaults already gave us a very good answer for an actual data set that we want to Carol. So, like the tweets are always you know the main piece there they're just sometimes they're totally necessary, but quite often you can go a long way without doing any food store. So today we're going to look at something I think may be even more important than building a predictive model, that's good at predicting, which is to learn how to interpret that model, to find out what it says about your data. To actually understand your data better by using machine learning - and this is kind of contrary to this - the common refrain that things like random forests are black boxes that hide meaning from us and you'll, see today that the truth is quite the opposite.

The truth is that random forests allow us to understand our data deeper and more quickly than traditional approaches. The other thing we got to learn today is how to look at larger data

sets than those which you can import with just the defaults and specifically we're going to look at a data set with over 100 million rows, which is the current kaggle competition for groceries. There anybody had any questions outside of those two areas since we're talking about today or comments Emily yeah. It is this kind of like basic just to make sure I'm understanding the concept. I'm talking here. Sorry um. Can you just talk a little about like in general? I understand the details more now: random florists but like when do you know this is an applicable model to use in general? Be like? Oh, I should try random forests here because that's the part that I'm still like yeah, if I'm told to I, can yeah so the short answer is. I can't really think of anything offhand that it's definitely not going to be at least somewhat useful for so it's always worth trying. I think really, the question is in what situations should I try other things as well, and the short answer to that question is for unstructured data. What I call unstructured data. So, where are all the different data points represent the same kind of thing like a wave form in a sound or speech, or the words and piece of text or the pixels in an image? Are you almost certainly you're going to want to try deep learning and then outside of those two there's a particular type of model? We're going to look at today called a collaborative filtering model where which it so happens that the groceries competition is at that kind.

Where neither of those approaches are quite what you want without some tweaks to them, so that would be the other main one. So you're saying neither using deep learning and neither deep learning or random forests is exactly what you're wanting in to kind of do some quests. Yes, yeah, if anybody thinks of other places where maybe neither of those techniques is the right thing to use, yeah mention it on the forums, even if you're, not sure you know, so we can talk about it, because I think this is one of the more interesting Questions and to some extent it is a case of practice and experience, but I do think there are, you know true main classes. No, no. So last week we at the point where we had kind of done some of the key steps you know like over CSV reading, in particular, which door you know a minute or two at the end of that we saved it to a feather format, file and just To remind you, that's because this is basically almost the same format that it lives in

2. 00:05:10

- **dealing with missing values present in Test but not Train (or vice-versa) in 'proc_df()' with "nas" dictionary whose keys are names of columns with missing values, and the values are the medians.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

In RAM, so it's like ridiculously fast to read it and write stuff from from feather point. So what we're going to do today is we're going to look at lesson. 2, RF interpretation and the first thing we're going to do is read that feather format file. Now one thing to mention is a couple of you pointed out during the week a really interesting little little bug or little issue, which is in the proc DF function. The proc DF function remember, finds the numeric columns, which have missing values and creates an additional boolean column, as well as replacing the messing with medians and also turns the categorical objects you know into into the integer codes, the main things it does and coming. You pointed out some key points about the missing value handler. The first one is that your test set may have missing values in some columns that weren't in your training set, or vice versa, and if that happens, you're going to get an error when you try to do the random forest, because it's going to say you know, If that

is missing, field appeared in your training set, but not in your test set that ended up in the model. It's going to say you can't use that data set with this model, because you're missing one of the columns it requires. That's problem number one problem number two is that the median of the missing belt are the median of the numeric values in the test.

Set may be different for the training set, and so it may actually process it into something which has different semantics. So I thought that was a really interesting point. So what I did was I changed property air. So it returns a third thing na s and the na s thing it returns. It doesn't matter in detail what it is. But I'll tell you to say you know. That's a dictionary that, where the keys are the names of the columns that had missing values and the values of the dictionary are the medians and so then optionally. You can pass n A's as an additional argument to prop D F and it will make sure that it adds those specific columns and it uses those specific medians. Okay. So it kind of it's it's giving you the ability to say, process this test set in exactly the same way as we process this training center. Can you pass? That is a did. It features we just yeah, so I just did that like it's just a good pool, yeah in fact, that's a good point before you start doing work any day. I would start doing it get pull and it something's not working today that was was working yesterday check the forum, where they'll be an explanation of why you know this. This library, in particular, is moving fast, but pretty much all the libraries that we use, including pytorch and particular move fast, and so one of the things to do if you're, watching this on through the MOOC, is to make sure that you go to course doc. Bastard AI and check the links there because they'll be links saying.

Oh, these are the differences from the course so they're kind of kept up to date so that you're never gon na, because I can't edit what I'm saying yeah but yeah. Do it get Paul before you start each day, so I haven't actually updated all of the notebooks to add the extra return value i will over the next couple of days, but if you're using them, you'll just need to put next to a comma, and it is Yeah, otherwise we're going to error them its return through things, and you only have room for two things: okay, what I want to do, I think what I want to do before I talk about interpretation is to show you what the exact same process.

3. 00:09:30

- **Starting Favorita notebook,**
- **The ability to explain the goal of a Kaggle competition or a project,**
- **What are independent and dependant variables ?**
- **Star schema warehouse database, snowflake schema**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Looks like when you're working with a really large data set so and you'll see it's kind of almost the same thing. But there's going to be a few cases where we can't use the defaults, because the default kind of flag just run a little bit too slowly. Right so specifically, I'm going to look at the travel groceries, competition, specific lady. What's a call here, it is compress your favorite grocery sales forecasting. So this competition, Oh who's, who who was entering this competition? Okay, lot of you who would like to have a go at explaining what this competition involves, what what the data is when family, okay, trying to predict the items on the shelf, depending on lots of factors like oil prices, let me say, predicting the items on the Show what do you mean? What are you actually predicting? How much change to help me stop to maximize their? I guess it's not quite what we're predicting, but not trying to fix that

yeah and then there's a bunch of different data sets that you can use to do that. There's oil prices there's a stores, there's locations and each of those can be used to try to predicted okay. Does anybody want to have a go at expanding on that all right? So we have a bunch of information on different products, so we have so forth all right so far, every stool for every item.

For every day, we have a lot of related information available like the location where the school was located, the class of the product and units soared and then based on this, we are supposed to forecast in a much shorter timeframe compared to the training data. For every item number how much we think it's going to sell so only the units and nothing else so um. So your ability to explain the problem you're working on is really really important. Okay, so if you don't currently feel confident of your ability to do that, practice right with someone who is not in this competition tell them all about it. So in this case, but or in any case really, the key things to understand a machine learning problem would be to say what are the independent variables and what is the dependent variable. So the dependent variable is the theme that you're trying to predict. The thing you're trying to predict is how many units of each kind of product were sold in each store on each day during the two-week period. So that's the thing that you're trying to predict and the information you have to predict. It is how many units of each product at each store on each day were sold in the last few years and for each store it's a metadata about it like. Where is it located and what classes or is it for each type of product? You have some metadata about it, such as what category of product is it and so forth for each date we have some metadata about it, such as what was the oil price on that date, so this is what we would call a relational data set.

So a relational data set is one where we have a number of different pieces information that we can join together. Specifically, this kind of relational database data set is what we would refer to as a star schema. A star schema is a kind of data, warehousing schema where we basically say there's some central transactions table. Then this place the central transactions table we go in the data section here is train dot, CSV and it contains the event number of units that were sold by date by store ID by item ID okay. So that's the central transaction state, where it's more very simple and then from that we can join various bits of metadata and it's called a standard schema because you can kind of imagine the transactions take on the middle and then all these different metadata tables join onto It giving you more information about the date, the item ID and the store already. Okay, sometimes you'll also see a snowflake schema, which means they might then be additional information joined on to maybe the the items table. That tells you about different item categories and store to the store table coming about the stage that the stores in and so for, wholesomely. Okay. So that's the basic information about this problem, the independent variables, the dependent variable - and you probably also wanting about like things like the time frame. Okay, now we start in exactly the same way as we did before loading in exactly the same spot setting the path.

But when we go to read CSV, if you say limit memory equals false right, then you're, basically saying use as much memory as

4. 00:15:30

- **Use dtypes to read data without 'low_memory = False'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

You like to figure out what kinds of data is here: it's going to run out of memory pretty much regardless of how much memory you have. So what we do in order to limit the amount of space that it takes up and read it in is we create a dictionary for each column, name through the data type of that column right and so for you to create this. It's basically up to you to you, know, run less or head or whatever, on the data set to see what the types are and to figure that out and pass them in. So then you can just pass in D type equals with that dictionary and so check. This out right, we can read in the whole CSV file in one minute and 48 seconds, and there are one hundred and twenty five point: five million roads so like whence people say like pythons, slow, now, pythons, not slow, I think, can be slow. If you don't use it right, but we can actually cause a hundred and twenty-five million CSV records in less than two minutes my language had on for just a moment. Actually, if it's fast almost certainly it's going to see yeah. So Python is a wrapper around a bunch of C code, usually yeah, so yeah, so Python itself isn't actually very fast yeah, so that was Terrence paw, who writes things for writing programming languages for a living so and he's right itself is not fast, but almost everything We want to do in Python, and data science has been written for us in C or actually more often in scythe on which is a like language which compiles to C, and so most of the stuff we run in Python is actually running, not just the C Code, but actually in pandas, a lot of it's written in like assembly language, it's heavily optimized behind the scenes.

A lot of that is going back to actually calling for train back libraries for linear algebra, so there's layers one layer of speeds that actually allow us to spend less than two minutes. Reading that much data yeah. If we wrote our own CSV reader in pure Python, it would take. It takes thousands of times at least thousands of times longer than the optimized versions yeah. So for us, what we care about is the speed we can get in practice, and so this is pretty cool. We, as well as telling it what the different data types were. We also have to tell it as before, which things do you want to pass as as dense? That's I've noticed searching this dictionary of specifying 60 1433 ending date. I was wondering in practice: is it like faster, if you all specify them to endure slower or like any performance consideration, so the key performance consideration here was to use the smallest number of bits that I could to fully represent the column. So if I had used in 8 for item number, there are more than 255 biomass and but the I mean who was specifically the maximum item, number is bigger than 255. So, on the other hand, if I'd used in 64, the score number it's using more, it's necessary, given that the whole purpose here was to avoid running out of RAM, we don't want to be using up eight times more memory than necessary, so the key thing was Really about memory, and in fact, when you're working with large data sets very often you'll find the slope piece is the actually reading and writing program, not the actual CPU operations.

So very often that's the key performance consideration. Also, however, as a rule of thumb, smaller data types often will run faster and particularly if you can use Cindy so there's a single instruction, multiple data, a vectorized code. It can pack more more numbers into a single vector to run at once. Think that was all definitely simplified exactly right, but once you do this, the shuffle thing beforehand, I need anymore. They made assigned a random substitution yeah. So so, although here I've read in the whole thing, when I start, I never start by reading in the whole thing. So if you

5. 00:20:30

- **Use '`shuf`' to read a sample of large dataset at start**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to

Search the forum fish fish off shuf you'll find some tips about how to use this UNIX command to get a random sample of data at the command prompt. And then you can just read that, and the nice thing is that that way, like that's a good way, for example, to find out what data types to use, it is to read in a random sample and let pandas figure it out for you, yeah and in General, I do as much work as possible on a sample until I feel confident that I understand the sample before I move on so yeah. Having said that, what we're about to learn is some techniques for running models on this full data set that I'm actually going to work on arbitrarily large data sets, but also, I specifically wanted to talk about how to really invite analysis. One thing to mention: unpromoted object: objects are like like saying, create a general-purpose Python data type, which is slow and memory heavy, and the reason for that is that this is a boolean which also has missing values, and so we need to deal with this before we Can turn it into a boolean, so you can see after that. I then go ahead and I say, fill in the missing values with false. Now you wouldn't just do this without doing some checking ahead of time, but some exploratory data analysis shows that it seems that this is probably an appropriate thing to do. It seems that missing doesn't mean most.

It objects generally reading the string so replace for Strings. True and false, with actual volumes and then finally convert it to an actual boolean flag. So at this point, when I save this this file now over 123 million records takes up something under two and a half gigabytes of memory. So like that, you can do like run, you know, look at pretty large data sets even on pretty small computers, which is interests so at that point now that it's in a nice fast four-minute, look how fast it is. I can save it to feather format in under five seconds: okay, so that's nice and then because pandas is generally pretty fast. You can do stuff, like summarize, every column of all 125 million records in 20 seconds. Okay. So the first thing I looked at here actually is the dates right generally speaking, dates are just going to be really important and one of the stuff you do, particularly because any model that you put in in in practice you're going to be putting it in at Some note that is later than the date that you trained it by definition right and so, if anything in the world changes, you need to know how your predictive accuracy changes as well, and so what you'll see on cable and what you should always do in your Own projects is make sure that your plates don't overlap. So in this case, the dates that we have in the training set go from 2013 to mid August 2017. Okay, there's our first and last and then in our test set.

They go from one day later right August, the 16th until the end of the month. So this is a key thing that, like you, can't really do any useful machine learning. Until you understand this basic piece here, which is, you've, got four years of data and you're trying to predict the next two weeks. Okay, so that's just a fundamental thing that you're going to need to understand before you can go and do a good job at this, and so as soon as I see that what does that say to you, if you want it to now use a smaller data Set should you use a random sample, or is there something better you do probably from the bottom more recent yeah, okay, the most recent right and and if you ever have trouble answering questions like this just try to make it as physical as possible. So it's like! Okay, I'm going to go to a shop next week, and I am I've got a $ 5 bet with my brother as to whether I can guess how many cans of coke are going to be on the shelf all right. Well, probably, the best way to do that would be to go to the shop same day of the previous week and see how many cans of coke are on the shelf and guess it's going to be the same. You wouldn't go and look at family were there. Four years ago, but couldn't four years ago that same time frame of the year be important, I mean like, for example, how much coke they have in the shop at Christmas time is gon na be way more than so exactly so it's not

that. There's no useful information from four years ago, and so we don't want to entirely throw it away, but as a as a first step like what was this? What's the simplest possible thing, it's kind of like submitting the means, I wouldn't submit the mean of 2012 sales.

I would want to probably submit the mean of last month's cents, so yeah we just want to think about like how might we want to kind of create some initial easy models and how and later on like we might want to weight it. So, for example, we might want to wait for recent dates, more highly they're, probably more relevant, but we should do a whole bunch of exploratory data analysis to check that. So here's what the bottom of that data set looks like okay and you can see literally it's got a date, a store, number and item number and give it sales and tells you whether or not that particular item was on sale at that particular store on that Particular day and then there's some Terry ID right, so that's it so now that we have read that in we can do stuff like this is interesting again we have

6. 00:26:30

- **Take the Log of the sales with 'np.log1p()',**
- **Apply 'add_datepart)',**
- **'split_vals(a,n)',**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

To take the log of the sales and it's the same reason as we looked at last week, right because we're trying to predict something that kind of varies according to ratios. They told us in this in this competition that the root mean squared log error is something they care about, so we take a look they mentioned. Also, if you check the competition details, which should always should read carefully the definition of any project, you do it's, they say that there are some negative sales that represent returns and they tell us that we should consider them to be zero for the purpose of this Competition, so I clip the sales so that they fall between zero and no particular maximum. It's a clip just means cuddle after that point, truncate it and then take the log of that plus one. Why do I do plus one? Because again, if you check the details of the cable competition, that's what they tell you, they're going to use is they're not actually just taking the root mean squared log error, but the root mean squared log plus one there. Okay, because log of zero doesn't make sense. We can add the date part as usual, and you know again it's taking a couple of minutes right, so I would run through all this on a sample first. So everything takes ten seconds to make sure it works just to check everything.

That's reasonable. Before I go back so I don't want to wait two minutes or something I don't know what's going to work, but as you can see all this, all these lines of code are identical to what we saw for the bulldozers competition in this case and all I'm Bringing in is a training set. I didn't need to run trained cats because all of my data types are already numeric. Okay, if they weren't, I would need to call trained cast and then I would need to call the ply cats to apply the same categorical codes that I down from the training set to the validation set. I call prop D F as before, to check

7. 00:28:30

- **Models,**
- **'set_rf_samples',**

- **'np.array(trn, dtype=np.float32',**
- **Use '%prun' to find lines of code that takes a long time to run**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

The missing values and so forth, so all of those lines of code are identical. These lines of code again are identical because root mean square errors, store we care about, and then I've got two changes. The first is sent our F samples, which we learnt about last week, so we've got 120 something million records. We probably don't want to create a tree from our hundred twenty million, something reckless. I don't even know how long that's going to take. I haven't been. I haven't at the time and patience for wagon see so you know you can start with ten thousand or a hundred thousand, you know maybe runs in a few seconds, make sure it works, and you can kind of figure out how much you can run, and so I found getting it to a million, it runs in under a minute alright, and so the point here is: there's no relationship between the size of the data set and how long it takes to build the random forests. Their relationship is between the number of estimators multiplied by the sample size. Okay, I'm just curious what Angela's cousin has always been negative one yeah, so the number of jobs is the number of cause. Thursdaya news - and I was running this on a computer that has about 60 cause - and I just found if you try to use all of them, had spent so much time spending I've drops, it was doing slower.

So if you've got like lots and lots of cores on your computer, sometimes you want less than negative one means use every single core and there's one more change I made, which is that I converted the data frame into an array of floats and then I fit It on that, why did I do that, because, internally inside the random forest code, they do that anyway, right and so, given that I want to run a few different random forests with a few different hyper crepitus by doing it once myself, I save that minute 37 Seconds right, so if you run a line of code at a text like quite a long time so the first time I ran this random first regressor, a kind of took two or three minutes, and I thought I don't really want to wait to a few minutes. You can always add in front of the line of code: hey run a percent P right and what percent p run does is? It runs something called a profile and what a profiler does is it'll. Tell you which lines of code behind the teens took the most time right, and in this case I noticed that there was a line of code inside scikit-learn. That was this line of code and it was taking all the time the other day. And so I thought. Oh I'll do that first and then I'll pass you the result, and I won't do it again. Okay, so this thing of looking to see which things is taking up the time is called profiling and in software engineering is one of the most important tools you have data scientists really under appreciate this tool that you'll find like amongst conversations on issues or on Twitter Or whatever, I'm at the top data scientists, they're sharing and talking about profilers all the time and that's how easy it is to get a profile so for fun.

You know try running peer on from time to time on stuff, that's taking 10 20 seconds and see if you can learn to interpret and use profile outputs. You know, even though, in this case I didn't write this scikit-learn plus, I was still able to use the profile to figure out how to make it run over twice as fast, by avoiding recalculating this each time. So in this case I build my regressor. I decided to use 20 estimators. Something else that I noticed in the profiler is that I can't use our base, for when I do is set our f samples, because if I do it's going to use the other 124 million rows to calculate the oeob score, which is like again, it's still going To take forever so I may as well have a proper validation set anyway, besides, which I want the validation set. That's the most recent dates, rather than is random. So if you use set RF

samples on a large data, set, don't put the low B score parameter in because it takes forever. So that got me a point. Seven six validation root mean squared log error and then I tried like fiddling around a different min samples. So if I decrease the min sample say from 100 to 10, it took a little bit more time to run as we'd expect and the arrow went down from 76 to 71. So that looks pretty good, so I kept decreasing it down to 3 and that brought

8. 00:33:30

- **We only get reasonable results, but nothing great on the leaderboard: WHY ?**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

This arrow down to 0.7, oh and when I decrease it down to 1, we didn't really know so. I kind of had like a reasonable random forest yeah. When I say reasonable, though, it's not reasonable in the sense that it's it's does not give a good result on the way to one, and so this is a very interesting question about. Why is that? And the reason is really coming back to Savannah's question earlier, like where my random forests not work as well. Let's go back and look at the data. Okay, here's the entire data set that we won at the whole data set. Here's all the columns that we used so the columns that we have to predict with are they the date, the store number? The item, number and weather is unpredictable day of week day of month day of year, is corner, start, etc, etc. So if you think about it most of the insight you know around like how much of something you expect to sell tomorrow, it's likely to be very wrapped up in the details about like what, where is that store? What kind of things do they tend to sell at that store for that item? What category of item is, is it you know if it's like fresh bread, they might not sell much of it on Sundays, because on Sundays you know and fresh bread doesn't get made. We're obvious it's gasoline, maybe they're gon na sell a lot of gasoline because on Sundays, people go and go up there, half with a wick ahead right now, a random forest has no ability to do anything other than create a bunch of binary splits on things like They have week store number item number.

It doesn't know which one represents gasoline. It doesn't know which stores are in the center of the city versus which ones are out in the it doesn't know any of these things, so its ability to really understand what's going on is somewhat limited, so we're probably going to need to use the entire four Years of data to even get some useful insights, but then the students beside using the whole four years of data and one of the data we're using, is really old. So, interestingly, there's a cable kernel that points out that what you could do is just take. The last two weeks and take the average sales, the average sales by date by store number by item number and just submit that and if you just submit that you come about 30th, all right. So for those of you in the groceries kind of Terrace. As I come into a question, I think this may have tripped me up. Actually, I think you said dates store item. I think it's actually store item sales and then you mean across date oh yeah, you're right. It's a store item at one promotion. I know promotion, is you know if you do it, if you do it like date as well, you end up, so these each row represents basically like a cross tabulation of all of the sales on that date in that store for the item. So if you put date in there as well, there's only going to be one or two items being averaged in each of those cells, which is you know too much variation basically two spots: it doesn't give you a terrible result, but it's it's not xxx.

So so your job, if you are looking at this competition and we'll talk about this in the in the next class, is how do you start with that model and make it a little bit better all right, because

if you can, then by the time we made Up next, hopefully, you'll be above the top 30, because you know CAG will be in Kaggle. Lots of people have now taken this kernel and submitted it and they all have about the same score and the scores are ordered not just by score but by date submitted. So if you now submit this kernel, you're not going to be 30th because you're way down the list when it was submitted all right. But if you could retire a bit better you're going to be better than all of those people. So try and think of how can you make this a tiny bit better? I could you try to capture seasonality and trend effects by creating new columns like these are the average sales in the month of August. These are the average sales for this year yeah. I think that's a great idea, so the thing for you to think about is how to do that right and so like see. If you can see, if you can make it work, because there are details to get right, which I know Terrance has been working on. This for the last week and he's got almost crazy, like the details. Prez here the details are: are difficult, they're not difficult, like intellectually difficult, they're kind of difficult in the way that makes you like, when I head back your desk at to any air and like this is something to mention in general.

Is the coding you do? The machine learning is like it's incredibly frustrating and incredibly difficult, not difficult like technically but difficult like there. If you get a detail wrong much of the time, it's not going to give you an exception, it will just silently be slightly less good than it otherwise would have been right and if your own peril, at least you know, okay, well, I'm not doing as well As other people won't haggle right, but if you're not on cattle, you just don't know like you don't know if your company's model is like half as good as it could be, because you made a little mistake right. So that's why? One of the reasons why practicing on tackle now is great right because you're going to get practice in finding all of the ways in which you can infuriatingly screw things up and you'll. Be amazed like for me there's extraordinary array array of them, but, as you get to know what they are you'll start to know how to check for them as you go right, and so the only way like you should assume every button you press you're, going to Press the wrong button right and that's fine as long as you have a way to find out. Okay, so we'll talk about that more during the course, but unfortunately there isn't like a set of specific things. I can tell you to always: do you just always have to think like okay? What do I know about the results of this thing? I'm about to do I'll. Give you a really simple example: a really simple example: if you've actually created that that basic entry entry, where you do take the mean by date by store number by own promotion, right and you've like submitted it and you've, got a reasonable score.

And then you think you've got something: that's a little bit better and you do predictions for that. How about you don't create a scatter plot, showing the predictions of your average model on one axis versus the predictions of your new model on the other axis. You should see that they just about form a line right and if they don't, then that's a very strong suggestion that you screwed something up alright, so like that, be an example. Okay, can you pass that one to the end of that room possible instance once those? Yes, so your so for a problem like this, unlike the car insurance problem on taggle, where we don't wear, columns are unnamed, we know we know what the columns representing what they are. Do you? How often do you pull in data from other sources? To supplement that I mean you could maybe like weather data, or you know, for example, or how often is that used very often right, and so the whole point of this star schema is that you break the second table and then you've got these other tables coming On third, that provide metadata about it. So, for example, whether it's meditating about a date yeah on cattle, specifically most competitions, have the rule that you can use external data as long as you post on the forum that you're using it and then it's publicly available. But you have to check on a competition by competition basis. They will tell you outside of cattle.

You should always be looking or like what external data could I possible leverage here? Okay, so are we still talking about how to tweak this data set? If you wish? Well, I'm not familiar with the countries here, so maybe he's Ecuador Ecuador. So maybe I would Ecuador largest grocery chain. Ecuador is largest first chain. Maybe I would start looking for Ecuador's holidays and shopping holidays, maybe when they have a three-day weekend, or you know I've, actually that information is provided in this case and so in general. One way of tackling this kind of problem is to create lots and lots of new columns containing things like you know: average number of sales on holidays, average percent change in sale or between January and February, and so on and so forth. And so, if you have a look at there's been a previous competition on cattle Rossman store sales that was

9. 00:43:30

- **Quick look at Rossmann grocery competition winners,**
- **Looking at the choice of validation set with Favorita Leaderboard by Terence Parr (his @ pseudo here ?)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Almost identical it was in Germany in this case for a major grocery chain, how many items are sold by day by item type by store and the this case. The person who won quite unusually actually was something of a domain expert in this space they're. Actually, a specialist in doing logistics predictions - and this is basically what they did was professional sales forecast consultant ping. He created just lots and lots and lots of columns based on his experience of what kinds of things can be useful for making predictions to that. But that's an approach that can work. The third-place team did almost no feature engineering, however, and also they had one big oversight which I think they would have won. So you don't necessarily have to use this approach, so we're learning anyway, we'll be learning a lot more about how to win this competition and ones like as we go. They did it to you, the third question, so if you, google, for among cattle rustlin you'll, see it the short answer, is there is a big blow. So what are the things? And these are a couple of charts that so Terrence is actually my teammate on this competition. So parents drew a couple of these charts for us and I want to talk about this, which is, if you don't have a good validation set. It's it's hard, if not impossible, to create a good model, so in other words like if you're trying to predict next month's sales - and you build a bunch, you know you try to build a model and you have no way of really knowing whether the models you've Built are good at predicting sales a month ahead of time.

Then you have no way of knowing, when you put your model in production, whether it's actually going to be any good okay. So so you need a validation set that you know is reliable and telling you whether or not your model is likely to work well when you like, put it into production or use it on the test set. So in this case, what Terrance has plotted here is, and so normally you should not use your test set for anything other than using it right at the end of the competition all right end of the project to find out how you've got. But there's one thing: I'm going to let you use the test set for an addition, and that is to calibrate your validation set. So what Terrance did here was he built four different models right, some which he thought would be better than others, and he submitted each of the four models to cattle to find out its score, and so the x-axis is the score: the cattle produce on the leaderboard. Okay and then on the y-axis he plotted the score on a particular validation set. He was trying out to see whether this validation said look like it was going to be any good. So if your validation set is good,

then the relationship between the leaderboards for by the testing score and your validation set score should lie in a straight line. Ideally, it'll actually lie on the y equals x line, but honestly that doesn't matter too much as long as relatively speaking, it tells you which models are better than which other models, then you know which model is the best right, and you know how it's going to Perform on the test set, because you know the linear relationship between two things: okay, so in this case, Terrence has managed to come up with a validation set, which is looking like it's going to predict our tab or leaderboards for pretty well, and that's really cool right.

Because don't even go away and try a hundred different types of models, feature engineering waiting, twigs paper parameters, whatever else see how they go on the validation set and not have to submit to tackle right, so we're going to get a lot more iterations, one more feedback. This is not just true of cattle, but every machine learning project you do yeah, and so, if you find so here's a different one, he tried back where it wasn't as good right. It's like oh these ones that were quite close to each other. It's showing us! The opposite direction: that's a really bad sign. It's like okay, this validation set idea didn't seem like a good idea. This validation set idea did number one good idea. So in general, if your validation set, it's not showing a nice straight line, you need to think carefully. Like okay, how is the test set constructor, why? How is my validation set different in some way, you're constructing images which is different, but I have to draw lots of charts and so forth. So one question is, and I modified two to guess how how you did it so how they actually tried to construct this validation set us close today. So I would try to do is to try to sample points from the training set that are very closer possible to some of the points in the test set. First, in what sense - and I don't know I will have to find any lectures, but in this case for this first room 4030 scrotus.

The last points was trying variations off someplace right, so the most recent you know and what I noticed was so first I looked at the date range of the test set and then I looked at the the kernel that described how he or she here's the date Range of the last two weeks of August 26 - that's right, and then the person who submitted the kernel that said how to get the 0.58 leaderboard position or whatever sport everything. I looked at the date range of that, and that was well. There was actually 14 days and the test set is 16 days, but the interesting thing is the test set, begins on the day after pay day and ends on the pay day, and so these are things I also paid attention to, but - and I think that's one Of the bits of better data that they told us, you know, so these are the kinds of things you just put a like. Try and, like I said, trick plot lots of pictures and like even if you didn't know it was pay day. You know you would want to like draw the time series part of sales, and you would hopefully see that like every two weeks, so it'd be a spike or whatever you'd be like. Oh, I want to make sure that mine, I have the same number of spikes in my validation set that I've had in my test day. Okay, let's take a five-minute break and let's come back at 2:30 to

10. 00:50:30

- **Lesson2-rf interpretation,**
- **Why is 'nas' an input AND an output variable in 'proc_df()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, so um. This is my favorite bit of interpreting machine learning models by the way, if you're looking for my notebook about the groceries competition, you won't find it in github,

because I'm not allowed to share code for running competitions with you unless you're on the same team as me, After the competition is finished, it'll be on github. So if you're doing this for the video, you should be able to find it. So, let's start by reading in our feather file. So after the file is exactly the same as our CSV file. This Israel, Blue Book for bulldozers, competition, so we're trying to predict the sale price of every industrial equipment, adoption and so reading. The feather format, by all means that we've already read in the CSV and processed it into categories, and so the next thing we do is to run property F in order to turn the categories into integers deal with the missing values and pull out the hidden variable. Okay, this is exactly the same thing as we used last time to create a validation set where the validation set represents the last couple of weeks, the last twelve thousand records by date - and I discovered thanks to one of your excellent questions on the forum last week. I had a bug here, which is that proc Pierre was shuffling the order, sorry so doc, Proctor, yes, and last week we saw a particular version of property earth where we passed in a subset and when I passed in a subset it was randomly shuffling and so Then i said, split bowels: it wasn't getting the last rose by date, but it was getting a random set of roads.

So I've now fixed that. So if you really run the lesson, 1rf code you'll see slightly different results. Specifically, you'll see in that section that my validation set results, look less good, but that's only for this tiny little bit where I had subset equals. Yes, I'm a little bit confused about the notation here, so only as is most a input variable and this associate variable dysfunction and yeah. Why is thatis the proc DF returns a dictionary telling you, which things were missing, which columns are missing and for each of those columns what the median was. So when you call it on, like the larger data, set the non subset, you want to take that return value, and you don't pass in an energy to that weight. You just want to get back a result later on when he planted into a subset. You want to use to have the same missing columns and the same medians, and so you pass it in and if, like this different subsets like if it was a whole different data set turned out had some different missing columns. It would update that dictionary with some with additional key values as well. So it kind of you can you don't have to pass it in if you don't pass it in, but it just gives you gives you the information about what was missing and the medians. If you do a sit in, it uses that information for any missing columns that that are there and if there are some new missing columns that will update that dictionary with that, so it's like keeping our datasets common information yeah.

It's got a keep track of all any any missing columns that you came across in any of anything. You pass the property yeah. Thank you, okay, so we split it into the training and test set just like we did last week and so to remind you once we've done crop DF. This is what it looks like. This is the log of sale, price, okay. So the first thing to think about is we already know how to get the predictions right, which is we take the average. We take the average value in each leaf node in each tree after running a particular road through each tree. That's how we get them. The prediction, but normally we don't just want a prediction. We also want to know how confident we are of that prediction, and so we would be less confident of a prediction if we haven't seen many examples of roads like this one and if

11. 00:55:30

- **How confident are we in our predictions (based on tree variance) ?**
- **Using 'set_rf_samples()' again.**
- **'parallel_trees()' for multithreads parallel processing,**
- **EROPS, OROPS, Enclosure**

We haven't seen many examples of roads like this one. Then we wouldn't expect any of the trees kind of have a path through which which is really designed to help us predict that road and so conceptually. You would expect, then, that, as you pass this unusual row through different trees, it's going to going to end up in very different places. So in other words, rather than just taking the mean of the predictions of the trees and saying that's a prediction. What if we took the standard deviation of the predictions of the trees, so the standard deviation of the predictions of the trees. If that's high, that means each tree is giving us a very different estimate of this rose prediction. So if this was a really common kind of row right, then the trees follow forward to make good predictions for it, because it's same lots of opportunities to split based on those kinds of roads right. So the standard deviation of the predictions across the trees gives us some kind of at least relative understanding of how confident we are of this prediction. So that is not something which exists in scikit-learn or in any library I know of so we have to create it, but we already have almost the exact code. We need because remember last lesson: we actually manually calculated the averages across different sets of trees, so we can do exactly the same thing. There calculate the standard deviations, so we know I'm doing random forest interpretation.

I pretty much never use the full data set. I always call set our samples because, like we don't need a massively accurate, random forest, we just need one which indicates the nature of relationships involved right, and so I just make sure this number is high enough that if I call the same interpretation commands multiple times, I don't get different results back each time. That's like the rule of thumb about how big does it need to be right, but in practice, like 50,000, is an odd number and most of the time you know it would be surprising if that wasn't enough and it runs in seconds. So I generally start with 50,000. So with my 50,000 samples per tree set, I create 40 estimators. I know from last time that min samples we people three max features. It cost point. Five, isn't bad and again we're not trying to create the world's most predictive tree. Anyway, so that all sounds fine, we get an r-squared on the validation set of 0.89. Again we don't particularly care, but it's long as it's good enough, which it certainly is, and so here's where we can do that exact same list. Comprehension, as last time remember, go through each estimator at each tree. Call drop predict on it, with our validation set, make that a list comprehension and pass that to NP stack, which in catenate s -- everything in that list across a new axis.

Okay. So now our rows are the results of each tree and their columns are the result of each row in the original data set. And then we remember we can calculate the mean right. So here's the prediction for a data set row number one and here's our standard deviation. Okay, so here's kind of do it for just one observation right at the end here, we've calculated for all of them, just pretty nipple one here now this this can take quite a while, and specifically it's not taking advantage of the fact that my computer has lots Of cause in it list comprehensions, this is this is like the list. Comprehension itself is Python code that it's my Python code and python code. Unless you're doing special stuff runs in serial, which means it runs on a single CPU doesn't take advantage of your multi CPU hardware, and so, if I wanted to run this, you know on more trees and more data. You know this. One second is going to go out and you see here the wall time. The amount of actual climate talk is roughly equal to the CPU time. Where else, if it was running on lots of cause, the CPU time would be higher than the war time. So it turns out that scikit-learn provides a handy national circuit load fastai provides a handy function called parallel trees, which pulls some stuff inside cyclone and parallel. Trees

takes two things: it takes a random forest model that I train the here.

It is and and some function to call and it calls that function on every tree in parallel, so in other words, rather than calling T, I predict X ballad, let's create a function that calls T dot, predict X, salad, let's use parallel trees to call it on Our model to every tree, okay and it will return a list of the result of applying that function to every tree. And so then we can NP Don stack that so, hopefully you can see that that code and that code are basically the same thing right. But this one is doing it in parallel, and so you can see here now our wall time has gone down to 500 milliseconds, and it's now giving this exactly the same answer. Okay, so a little bit faster time committing we'll talk about more general ways of writing code that runs in parallel. That turns out to be super useful for data science, but here's one that we can use it's very specific to random forests. Okay, so what we can now do is we can always call this to get our predictions for each tree and then we can call standard deviation to then get them for every row, and so let's try using that. So what I could do is, let's carried a copy of our data and, let's add an additional column to it, which is the standard deviation of the predictions across the first axis. Okay and, let's also add in the name so they're the predictions themselves. So we, you might remember from last lesson that one of the predictors we have is called enclosure and we'll see later on.

But this is an important predictor and so, let's start by just doing a histogram. So one of the nice things and panders is: it's got built in body and capabilities. It's well worth googling for pandas plotting to see how yes, Terrence Jeremy. Can you remind me what enclosure is so we don't know what means, and it doesn't matter. You know, like. That's the whole purpose of this process is that we're going to pick it out we're going to learn about what things are, or at least what things are important and what later on figure out what they are and how they're important so we're going to start out. Knowing nothing about this data set all right, so there's something so I'm just going to look at something called enclosure that has something called erupts and something called robots, and I don't even know what this is yet. All I know is that the only three that really appear at any great quantity are over ops erupts, whe and eros, and this is like really common. As a data scientist, you know you often find yourself looking at data that you're not that familiar with and you've got to figure out at baseline, which fits to study more carefully and which gets into matter and so forth. So, in this case at least know that these three groups - I really don't care about because they basically don't exist so given that we're going to ignore those three, so we're going to focus on this one here this one here and this one here, and so here You can see what I've done is I've taken my data frame and I've grouped by enclosure, and I am taking the average of these three fields.

So here you can see here's the average sale price, the average prediction and the standard deviation of prediction. For each of my three groups, so I can already start to learn a bit here, as you would expect the prediction and the sale price are close to each other on average. So that's a good sign and then the standard deviation varies a little bit. It's a little hard to see in a table, so what we could do is we could try to start like printing these things out. So here we've got the sale price or each little of enclosure, and here we've got the prediction: peach level of enclosure and for the error bars I'm using the standard deviation of perdition. Alright, so here you can see the actual and here's, the prediction and here's. My confidence in the world, okay, or at least it's the average of the standard deviation of the random forests. So this tells us it'll tell us if there's some groups or some rows that we're not very confident of it all. So we could do something similar for product size right, so here's different product sizes. We can do exactly the same thing of looking at our predictions under standard deviations. Okay, we could sort by and what we could say is like well. What's the ratio of the standard deviation of

the predictions to the predictions and selves right, so you kind of expect on average that when you're predicting something that's a bigger number that your standard deviation would be higher all right.

So you can like sort by that ratio, and what that tells us is that the product size, large and product size, compact, our predictions, are less accurate. You know as relatively speaking as a ratio of the total price, and so then, if we go back and have a look well there you go. That's why, for the histogram, those are the smallest groups. Okay, so, as you would expect in small groups, we're doing the less good job right, so this confidence interval you can really use for two main purposes. One is that you can group it up like this and look at the average confidence interval by group to find out. Are there some groups that you just don't seem to have confidence about about those groups, but perhaps more importantly, you can look at them for specific roads, and so, when you put it in production, you might always want to see the confidence interval. So if you're doing say credit scoring so deciding whether to give somebody alone, you probably want to see not only what's their level of risk. But how confident are we and if they want to borrow lots of money and we're not at all confident or about our ability to predict whether they'll pay it back, we might want to give them a small amount. Okay, so those

12. 01:07:15

- **Feature importance with 'rf_feat_importance()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Are the two ways in which you reduce this? Okay? Let me go to the next one, which is the most important. The most important is feature importance, and the only reason I didn't do this first is because I think the intuitive understanding of how to calculate confidence interval is the easiest one to understand entry. In fact, it's almost identical to something: we've already happened right, but in terms of which one do I look at first in practice, I always look at this in Baptist, so when I'm working on whether it be a candle competition or a real world project, I build A random forest as fast as I can try and get it to the point that it's led, you know significantly better than Brandon, but doesn't actually much better than that, and then the next thing I do is to plot the future importance and the future importance tells Us in this random forest, which columns mattered right, so we had like dozens and dozens but of columns originally in this data set - and here I'm just picking out the top ten, so you can just call our F feature importance again. This is part of the faster and ivory it's leveraging stuff. That's in scikit-learn parse in the model pass in the data frame is getting to know the names of columns right and it'll. Tell you it author, I'll, give you back a pandas data frame showing you in order of importance. How important was each column, and here on this note, pick out the top ten.

So we can then plot that right so fi, because it's a data frame, we can use data frame, plotting mints, so here I've plotted all of the future importances right, and so you can see here like and I haven't been able to write all of the names Of the columns at the bottom, which that's not the important thing, the important thing is to see that some columns are really really important and most columns don't really matter at all and like in nearly every data set you use in real life. This is what your feature importance is going to look like it's going to say: there's like a handful of columns that you care about, and this is why I always start here right because at this point, in terms of like looking into learning about this domain, heavy Industrial

equipment options I only got to care about learning about the columns which matter right so are we going to bother learning about enclosure depends whether enclosure is important and there it is it's in the top ten. So we are going to have to learn about enclosure. Okay, so then we could also plot this as a bar plot. All right, so you can hear I've just created a little little tiny little function here, that's going to just plot my bars and I'm just going to do it for the top 30, and so you can see the same basic shape here and I can see there's My enclosure, okay, so we're going to learn about how this is calculated in just a moment, but before you worry about how it's calculated much more important is to know what to do with it.

So the most important thing to do with it is to now sit down with your client or your data dictionary or whatever your source of information is and say to them. Okay, tell me about you made: what does that mean? Where does it come from plot lots of things like histogram Devere made and scatter plots of here made against price and learn everything you can because you're made and cupola system they're the things that matter right and what will often happen in real-world projects? Is that you'll sit with the client and you'll say? Oh it turns out. The capital system is the second most important thing, and then they might say that makes no sense. Now that doesn't mean that there's a problem with your model, it means there's a problem with their understanding of the data that they gave you okay. So let me give you an example: I entered a cattle competition where the goal was to predict which applications for grants at a university would be successful, and I used this exact approach and I discovered a number of columns which were almost entirely predictive of the dependent Variable and specifically when I then looked to see in what way their predictive, it turned out whether they were missing or not, was basically the only thing that mattered in his data set and so later on. So I ended up winning that competition and I think a lot of it was thanks to this insight right and so later on what had happened.

It turns out that at that University there's, you know, there's an administrative burden to fill any other database and so for a lot of the grant applications. They don't fill in the database for the folks whose applications weren't accepted right so in other words, these missing values. In the data set were saying: okay, this grant

13. 01:12:15

- **Data leakage example,**
- **Colinearity**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Wasn't accepted because if it was accepted, then you know the admin folks are going to go in and type in that information. So this is what we call data leakage and data leakage means there's information in the data set that I was modeling with which the university wouldn't have had in real life at the point in time they were making a decision right. So when they're deciding you know which grant applications, should I like prioritize, they don't actually know which ones the admin staff or later I'm going to add information to, because it turns out that they've got accepted. I mean right, so one of the key things you'll find here is is data leakage, problems and that's a serious problem. You need to deal with. The other thing that will happen is you'll often find it's signs of collinearity, and I think that's what's happened here with kapwa system. I think Kapler system tells you whether or not a particular kind of heavy industrial equipment has a particular feature on it. But if it's not that kind of industrial

equipment at all, it will be empty, they'll be missing, and so capitalist system is really telling you whether or not it's a certain class of heavy industrial equipment. Now this is not linkage. This is actual information you actually have at the right time. It's just that like interpreting it, you have to be careful okay, so I would go through at least the top 10 or like kind of look for where the natural breakpoints are and really study.

These things carefully to make life easier for myself. What I tend to do is I try to throw some data away and see if that matters. So in this case I had a random forest which let's go and see how accurate it was. Point. Eight nine point: eight eight, nine. What I did was I said here: okay, let's go through our feature, importance, data frame and filter out those where the importance is greater than 0.005 thanks. 0.025. Two point: zero five is about here right, it's kind of like where they really flattened off all right. So let's just keep those, and so that gives us a list of 25 column names. And so then I say: okay, let's now create a new data frame view which just contains those 25 columns, Coles flipped bowels on it again decision to test and training set and create a new random forest. And, let's see what happens - and you can see here - the a square basically didn't change. Eight, nine one versus eight, eight nine. So it's actually increased a tiny bit right. I mean, generally speaking, removing redundant columns. Well, you know it shouldn't. Obviously it shouldn't make it worse. If it makes it worse, they weren't redundant after all, it might make it a little better because if you think about how we built these trees, when it's deciding what to split on you know it's, it's got less things to have to worry about trying. It's less often going to like accidentally find a crappy poem. So it's you know.

I've got a slightly better opportunity to create a slightly better tree with slightly less data, but it's still it's not going to change it by much, but it's going to make it a bit faster and it's going to. Let us focus on what matters. So if I rerun feature importance now, I've now got twenty five. Now. The key thing that's happened is that when you remove redundant columns is that you're also removing sources of collinearity right, in other words, two columns that might be related to each other? Now Collini arity doesn't make your random forests less predictive, but if you have two columns that are related to each other, you know this column is a little bit related to this column and this column is a strong driver of the dependent variable. Now, what's going to happen is that the importance is going to end up like kind of split between the two collinear columns, it's going to say like well both of those columns matter so kind of a split between the two. So, by removing some of those columns with very little impact, it makes your feature importance, block clearer, and so you can see here actually year made was pretty close to couple system before, but there must have been a bunch of things that are collinear with year made, Which makes perfect sense right, like old industrial equipment, wouldn't have had a bunch of kind of technical features that new ones would, for example.

So it's actually saying like okay, you made really really matters all right, so I trust this feature importance better. You know the predictive accuracy of the model is a tiny bit better, but this feature importance says a lot less familiarity to confuse us. So let's talk about how this works, it is actually really simple and not only is it really simple, it's a technique. You can use not just for random forests, but for basically any kind of machine learning model. And interestingly, almost no one knows that, like many people will tell you all this particular kind of model there's no way of like interpreting it and the most important interpretation of a model is knowing like which things are important and that's almost certainly not going to be True because there's technique, another teacher, she actually works to any kind of model. So here's what we're going to do we're going to take our data, set the bulldozers right and we've got this column, we're trying to predict right, which is price and then we've got all of our independent

variables. Okay, so here's an independent variable here year made right plus a whole bunch of other variables and remember we had after we did a bit of hemming. We have 25 independent variables. Okay, how do we figure out how important year made is well? We've got our whole random first, but and we can find out our predictive accuracy where so we're going to put all of these rows through our random forest and we're going to spit out some predictions right and we're going to compare them to the actual price. You get in this case, for example, our root mean squared error and our R squared, and we're going to call that, like it's a starting point right so now, let's do exactly the same thing, but let's take the year made column and randomly shuffle it so randomly Permute just that column, so now you made has exactly the same, like distribution is to follow same mean, standard deviation, but it's going to have no relationship as a dependent variable at all, because we totally randomly reorder them so before we might have found our R squared With point eight nine right and then after we shuffle ear made we check again and now it's like point eight.

Oh, that's poor got much worse when we destroyed that variable and it's like okay, let's try again, let's put your made back to how it was and this time let's take enclosure and shuttle that right and we find this time an enclosure. It's point at four and we can say okay, so the amount of decrease in our score for year made was 0.09 and the amount of decrease in our score for enclosure was 0.05 right, and this is going to and I'll feature importances for each one of our Columns, yes, wouldn't just excluding, let's say each column, I'm running running the random forests and checking the decay in the perform. Yes, so you could remove the column and train a whole new, random forest, but that's going to be really slow. Where else this way, we can keep our random forest and just test the predictive accuracy of it again alright. So this is nice and fast by comparison. In this case, we just have to rerun every row forward through the forest for each shuffle column, Jason, we're just basically doing predictions of exactly okay question. So if you want to do like multi clone area, would you do two of them and random shuffle? And then three of them ran in trouble. Yes, so I don't think you need multicollinearity. I think you mean looking for interaction effects yeah. So if you want to say which pairs of variables are most important, you could do exactly the same thing each pair in in turn.

In practice, there are better ways to do that because that's obviously computationally pretty expensive and so we're trying to find time to do that again. Okay, so we now have a model which is a little bit more accurate and is we've learned a lot more about it? so we're out of time, and so what I would suggest you try doing now before the next class for this bulldozes data set, it's like go through the top. I don't know five or ten predictors and try and learn what you can about how to draw plots and pandas and try to come back with, like some insights, about like what's the relationship between a year made and the dependent variable. What's the histogram of year made, you know, try and find you know some possible or like now that you know you inmates really important. Is there some noise in that column, which we could fix? There's some weird encodings in that column that we've fixed this idea. I had that maybe a couple system is there entirely because it's collinear with something else johner we might try and figure out, whether that's true or so. How would you do it fi product class desk? That brings alarm bells. For me, it sounds like it might be. A high cardinality, categorical variable: it might be something with lots and lots with levels because it sounds like it's like a model name so like go and have a look at that model name.

Does it have some order into it? Could you make it an ordinal variable to make it better? It doesn't have some kind of hierarchical structure in the stream that we can split it on like to create more sub columns. You don't ever think about this, you know and and so try and make

it so that you know by Tuesday when you come back, you've got some new. Ideally, you've got a better accuracy than what I just showed, because we found some new insights or at least that you can tell the class about some things. You've learnt about how heavy industrial equipment options work in practice. Okay right, so your Tuesday

Lesson 04

Outline

- Forecasting: Grocery Kaggle discussion, Parallel to Rossman stores
- Random Forests: Confidence based tree variance
- Random Forests: Feature Importance Intro
- Random Forests: Decoupled Shuffling

Video Timelines and Transcript

1. 00:00:04

- **How to deal with version control and notebooks ? Make a copy and rename it with “tmp-blablabla” so it’s hidden from Git Pull**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

All right welcome back something to mention somebody asked on the forums. Really good question was like: how do I deal with version control and notebooks? The question was something like every time I change the notebook. Jeremy goes and changes it on git and then I do a git pull and I end up with a conflict, and I love that and that's that happens a lot with notebooks, because notebooks behind the scenes at JSON files, which, like every time you run even a Cell without changing it, it updates that little number saying like what numbered cell this is, and so now suddenly there's a change and so trying to merge notebook changes is a nightmare. So my suggestion, I'd like a simple way to do it is, is when you're looking at some notebook like less than 200. If interpretation, you want to start playing around with this. The first thing I would do would be to go file, make a copy and then in the copy, say, file rename and give it a name that starts with TMP. That will hide it from get right, and so now you've got your own version of that workbook. That you can that you can play with okay, and so, if you now do a git pull and see that the original changed it won't conflict with yours, and you can now see there are two different versions. There are different ways of kind of dealing with this Jupiter notebook get problem like everybody has it one one is there are some hooks you can use it like remove all of the cell outputs before you commit to get, but in this case I actually want the Outputs to be in the repo, so you can read it on github and see it.

So it's a minor issue, but it's a it's something which catches everybody. Ah, yes before we move on to

2. 00:01:50

- **Summarize the relationship between hyperparameters in Random Forests, overfitting and colinearity.**
- **‘set_rf_samples()’, ‘oob_score = True’,**

- **'min_samples_leaf=' 8m45s,**
- **'max_features=' 12m15s**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Interpretation of the random forest model. I wonder if we could summarize the relationship between the hyper parameters on the random forest and its effect on you know, overfitting and dealing with collinearity and yeah. That sounds like a question born from experience. Absolutely so I got ta. Go back to lesson 1 RF, if you're ever unsure about where I am. You can always see my top here courses mly and lesson 1 on earth in terms of the hyper parameters that are interesting and I'm ignoring, I'm ignoring like pre-processing, but just the actual hyper parameters. The first one of interest, I would say, is the set RF samples command, which determines how many rows are in each sample, so in each tree, you're created from how many rows n each tree. So before we start a new tree, we either bootstrap a sample so sampling, with replacement from the whole thing or we pull out a subsample of a smaller number of rows, and then we build a tree from there so so step. One is: we've got our whole big data set and we grab a few rows at random from it and we turn them into a smaller data set and then from that we build a tree right. So that's the size of that is set our F samples. So when we change that size, let's say this originally had like a million rows and we said, set our F samples, twenty thousand right and then we're going to grow a tree from there, assuming that the tree remains kind of balanced as we grow it.

Can somebody tell me how many layers deep with this tree be and assuming we're growing it until every leaf is of size? One yes, log base 2 of 20,000 right, okay, so the the depth of the tree doesn't actually vary that much depending on the month, samples right, because it's it's related to the log of the size. Can somebody tell me at the very bottom so once we go all the way down to the bottom, how many leaf nodes would there be speak up? What what do you think right, because every single leaf node has a single thing in it? So we've got obviously a linear relationship between the number of leaf nodes in the size of the sample. So when you decrease the sample size, it means that there are less kind of final decisions that can be made right. So therefore, the tree is is going to be less rich in terms of what it can predict, because it's just making less different individual decisions, and it also is making less binary choices to get to those decisions. So therefore, setting RS samples lower is going to mean that you over fit less, but it also means that you're going to have a less accurate, individual tree model right and so remember the way Braman the inventor of random first described. This is that you're trying to do two things when you build a model when you build a model with bagging one.

Is that each individual tree or as SQL you say, each individual estimator is as accurate as possible right on the training set. So it's like each model is a strong predictive model, but then the across the estimators relation between them is as low as possible so that when you average them out together, you end up with something that generalizes. So by decreasing the set RF samples number. We are actually decreasing the power of the estimator and increasing the correlation, and so is that going to result in a better or a worse, validation set result for you. It depends right. This is the kind of compromise which you have to figure out when you do machine learning models. Can you pass that back there? If I wait, if I put the Bovie value equal to so it is basically dividing every 30. It ensures that the data won't be there in each three right now, our page. Second, probably if I put all the equal to two yeah random voice, yeah, so is it that make sure that out of my entire data 37 personal data would be there in every tree. So all our P equals true does. Is it says whatever your sub sample is, it might be a

bootstrap sample or it might be a subsample take all with the other rows right and put them into a tree tree and put them into a different data set and calculate the the error on those. So it doesn't actually impact training at all. It just gives you an additional metric, which is the oob error.

So if you don't have a validation set, then this allows you to get kind of a quasi validation set for free. If you want to set out a sample Aref sample so that the default is actually, if you say reset our F samples and that causes it to bootstrap, so it all sample our new data set as big as the original one, but with replacement okay. So obviously the second benefit of set our samples is that you can run more quickly and particularly, if you're running on a really large data set like a hundred million rows, you know it won't be possible to run it on the full data set. So you would either have to pick a subsample if yourself before you start or you set our examples. The second key parameter that we learnt about was min samples leaf. Okay. So if I changed min samples leaf for we assumed that men samples leaf was equal to 1, all right, if I set it equal to 2, then what would be my new depth? How deep would it be? Yes, log base to 20,000 minus one okay. So, each time we double the min samples leaf, we're removing one layer from the tree and fine I'll come back to you again since you're doing so well, how many leaf nodes would there be in that case, but how many leaf nodes would there be? In that case, 10,000, okay, so we're gon na be again dividing the number of leaf nodes by that number.

So the result of increasing min samples leaf is that now each of our leaf nodes has more than one thing in so we're going to get a more stable average that we're calculating in each tree. Okay, we've got a little bit less depth. Okay, we've got less decisions to make and we've got a smaller number of leaf nodes. So again we would expect the result of that would be that each estimator would be less predictive, but the estimators would be also less correlated so again. This might help us to avoid overfitting. Could you pass the microphone over here? Please, oh hi, Jimmy, I'm not sure. If, in that case, every node will have exactly two. No, it won't necessarily have exactly two, and I thank you for mentioning that, so it might try to do a split and so one reason well what would be an example, Chen XI that you wouldn't split, even if you had a hundred nodes. What might be a reason for that sorry, 100 items in a leaf, node they're, all the same they're, all the same in terms of the independent to saw the dependent and it has the dependent right now I mean I guess either, but much more likely would be The dependent, so if you get to a leaf node where every single one of them has the same option price or in classification like every single one of them, is a dog, then there is no split that you can do.

That's going to improve your information all right and remember, information is the term we use in a kind of a general sense in random for us to describe the amount of difference about at that additional information we create from a split is like: how much are we Improving the model so you'll often see this word, information gain, which means like how much better did the model get by adding an additional split point and it could be based on our MSC or it could be based on cross-entropy or it could be based on how Different to the standard deviations or whatever, so that's just a general term, okay, so that's the second thing that we can do again. It's going to speed up our training because it's like one less set of decisions to make remember, even though there's one less set of decisions. Those decisions like have as much data again as the previous set so like each layer of the tree can take like twice as long as the previous layer, so it could definitely speed up training and it could definitely make it generalize better. So then, the third one that we had was max features. Who wants to tell me what max features does I want to pass that back over there? Okay Vinay, we just leave their minds, how many features you're going to use in HP? In this case, it's a fraction up so you're

going to use up other, be just for each three, nearly right. What kind of right can you be more specific, or can somebody else be more specific? It's not exactly for each tree.

Can she that is that for each tree, randomly simple healthy? So not quite it's not for each tree, so the the set don't possibly care them. So the scent are of samples picks a picks, a subset of samples subset of rows for each tree, but min samples leaf. Sorry that max features doesn't quite do that. It's not something different at each set smell ant will yeah right. So it kind of sounds like a small difference, but it's actually quite a different way of thinking about it, which is we do our set our samples. So we pull out our sub sample or a bootstrap sample and that's kept for the whole tree and we have all of the columns in there right and then with max features equals 0.5 at each point. We then, at each split we'd, pick a different half of the features and then here what you could pick a different half of the features - and here we'll pick a different half of the features, and so the reason we do. That is because we want the trees to be as as rich as possible right so particularly like if you, if you were only doing a small number of trees like you, had only ten trees and you picked the same column set all the way through the tree. You're not really getting much variety and what kind of things are confined? Okay, so this this way, at least in theory, seems to be something which is going to give us a better set of trees, picking a different, random subset of features at every decision point. So the overall effective max features again, it's the same.

It's going to mean that the traits individual tree is probably going to be less accurate, but the trees are going to be more buried, and in particular here. This can be critical because, like imagine that you've got one feature, that's just super predictive. It's so predictive that, like every random subsample, you look at always starts out by splitting on that same feature, then the trees are going to be very similar in the sense like they all have the same initial split right, but there may be some other are interesting. Initial splits because they create different interactions of variables so by like half the time that feature won't even be available at the top of the tree. So half at least half the trees are going to have a different initial split. So it definitely can give us more variation and therefore again it can help us to create more generalized trees that have less correlation with each other, even though the individual trees probably won't be as predictive in practice. We actually looked at have a little picture of this. That, as as you add, more trees right, if you have max features equals none, that's going to use all the features every time. Right then, with like very very few trees. That can still give you a pretty good error, but, as you create more trees, it's not going to help as much because they're all pretty similar, because they're all trying every single variable. Where else, if you say max, features, equal square root or max pictures equals log.

Two then, as we add more estimators, we see improvements. Okay, so there's an interesting interaction between those two and this is from the SK loan Docs. This cool little chat. Okay, so then things which don't impact out our training at all and jobs simply says how many cpu, how many cause do we run on? Okay, so it'll make it faster up to a point, generally speaking, making this more than like, eight or so they may have diminishing returns. -1 says use all of your cause, so there's wrote, there's I don't know why the default is to only use one core. That's seems weird to me: you'll definitely get more performance by using more cause, because all of you have computers with more than one core nowadays and then our base core equals true, simply allows us to see the low B score. If you don't say that it doesn't calculate it, and particularly if you had set RF samples, pretty small compared to a big data set or B, is going to take forever to calculate, hopefully at some point we'll be able to fix the library. So that doesn't happen. There's no reason that need be that way, but right now, that's that's how the Bible place. Okay, our base, Kuhn, okay, basic parameters that we can change. There are

more that you can see in the docs or shift tab to have a look at them, but the ones you've seen are the ones that I've found useful to play with so feel free to play with others as well and, generally speaking, you know max Features, as I said, max features are like either non means all of them about 0.5 or square root or log.

You know kind of those trees seem to work pretty well and then some in samples leaf. You know I would generally try kind of 1. 3. 5. 10. 25. You know 100 and like, as you start doing, that if you notice by the time you get to 10, it's already getting worse, then there's no point going further. If you get to 100, it's still going better, then you can keep trying right but they're. The kind of general amounts that most things in to sit in all right,

3. 00:18:50

- **Random Forest Interpretation lesson2-rf_interpretation,**
- **'rf_feat_importance()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So random forest interpretation is something which you could use to create some really cool cattle kernels. Now, obviously, one issue is the faster. I library is not available in Cabell kernels, but if you look inside fastai dot, structured right, remember you can just use double question. Mark to look at the source code for something or you can go into the editor to have a look at it, you'll see that most of the methods we're using or a small number of lines of code in this library and have no dependencies on anything. So you could just copy that little if you need to use one of those functions, just copy it into your kernel and and if you do to say this is from the first day a library you can link to it on github, because it's available on github. As open-source, but you don't need to import the whole thing right, so this is a cool trick. Is that because you're, the first people to learn how to use these tools, you couldn't start to show things that other people haven't seen right. So, for example, this confidence based on tree variance is something which doesn't exist anywhere else feature importance, definitely does, and that's already in quite a lot of cable kernels. If you're, looking at a competition or a data set that where nobody's done, feature importance being the first person to do, that is always going to win lots of votes because it's like the most important thing is like which features are important.

So last time we, let's just make sure we put our tree data, so we need to change this to add one extra thing, all right, so that's no load, no data, yet there's that data okay. So, as I mentioned, when we do mobile interpretation, I tend to set our of samples to some subset, something small enough that I can ran a model in under 10 seconds or so because there's just no point run running a super accurate model. Fifty thousand is more than enough to see you'll basically see each time you run an interpretation, you'll get the same results back, and so as long as that's true, then you you're already using enough data. Okay, so feature importance. We learnt it works by randomly shuffling a column, each column, one at a time and then seeing how accurate the model the pre trained model the model we've already built is when you pass it in all the data as before, but with one column shuffled. So some of the questions I got after class kind of reminded me that it's very easy to under appreciated and kind of magic. This approach is, and so to explain I'll mention a couple of the questions that I heard, and so one question was like: why don't we or what, if we just um, we took one column at a time and created a tree on just each one column at A time so we've got our data set, it's got a bunch

of columns, so why don't we just like? We have that column and just build a tree from that right and then, like we'll see which which columns tree is the most predictive.

Can anybody tell me why what why that may give misleading results about feature importance? Okay, we're going to lose the interactions between the features yeah. If we just shuffle them, it will be a bad randomness and we were able to both capture the interactions and the importance of the future. It's great yeah and - and so this issue of interactions is not a minor detail. It's like it's massively important so like think about this bulldozes data set where, for example, where there's one field called year made and there's one field called sale date and like if we think about it, it's pretty obvious that what matters is the combination of these two, Which, in other words, is like how old is the piece of equipment when it got sold? So if we only included one of these, we're going to massively underestimate how important that feature is now here's a really important point, though, if you it's pretty much always possible to create a simple like logistic regression, which is as good as pretty much any random first, If you know ahead of time exactly what variables you need exactly how they interact exactly how they need to be transformed and so forth right. So in this case, for example, we could have created a new field which was equal to year made as a sale date or sale year year made and we could have fed that to a model and got you know, got that interaction for us. But the point is we never know that, like you never like, you might have a guess of it.

I think some of these things are interacted in this way and I think this thing we need to take the log and so forth, but you know the truth. Is that the way the world works, the causal structures? You know, they've got many many things interacting in many many subtle ways right and so that's why using trees, whether it be gradient, boosting machines or random forests works so well. So can you pass that to Terrance? Please one thing that bit me years ago was also: I tried that doing one variable at a time thinking oh well I'll figure out which one's most correlated with the dependent variable, but what it doesn't pull apart is that what, if all variables are basically copied? The same variable then they're all going to seem equally important, but in fact it's really just one factor yeah - and that's also true here. So if we had like a column, appeared twice right, then shuffling that column isn't going to make the model much worse. Right there'll be, if you think about like how it was built some of the times, particularly if we had like max features is 0.5 and some of the times we're going to get version a of the column, some of the time to get going to get version B of the column, so, like half the time shuffling version, a of the column is going to make a tree a bit worse. Half the time it's going to make.

You know column B, you'll, make it a bit worse and so it'll show that both of those features are somewhat important and it'll kind of like share the importance between the two features, and so this is why a reco linearity, but collinearity literally means that they're linearly Related, so this isn't quite right, but this is why, having two variables that are related closely related to each other or more variables that are closely related to each other means that you will often underestimate their importance using this. This random first technique, um, yes, Terrence, and so once we've shuffled and we get a new model. What exactly are the units of these importances? Is this a change in the R squared yeah? I mean it depends on the library we're using. So the units are kind of like I never think about them. I kind of know that, like in this particular library, you know 0.005 is often kind of a cutoff

4. 00:26:50

- **'to_keep = fi[fi.imp>0.005]' to remove less important features,**

- **high cardinality variables 29m45s,**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

I would tend to use, but all I actually care about is is this picture right, which is the feature importance, ordered for each variable and then kind of zooming in turning into a bar plot and I'm kind of like okay, you know here, they're all pretty flat And I can see: okay, that's about 0.05, and so I removed them at that point and just see like the model. Hopefully the validation score didn't get worse and if it did get worse, I'll just increase this a little bit. So I decrease this a little bit until it. It doesn't get worse, so yeah. The units of measure of this don't matter too much and we'll learn later about a second way of doing variable importance. By the way, can you pass that over there is one of the goals here to remove variables that I guess your state, your score will not get worse. If you remove them, so you might as well get rid of them yeah. So that's what we're going to do next, so so what having looked at our feature importance plot, we said: okay, it looks like the ones like less than 0.005. You know that kind of this long tail of boringness. So I said: let's try removing them right. So, let's just try grabbing the columns where it's greater than 0.005, and I said, let's create a new data frame called DF. Keep which is DF train with just those cap. Columns create a new training and validation set with just those columns created a new random forest, and I looked see how the validation set score and the validation set.

Our MSC changed and I found they got a tiny bit better. So if they're about the same or a tiny bit better than the thinking, my thinking is well. This is just as good a model, but it's now simpler and so now, when I redo the feature importance, there's less collinearity right and so in this case I saw that year. Maid went from being like quite a bit better than the next best thing, which was couple system way better than the next best thing. Okay and coupler system went from being like quite a bit more important than the next two equally important to the next two. So it did seem to definitely change these feature importances and hopefully give me some more insight there. So how did that help our model in general? Look, what does it mean that your maid is no way yeah, so we're going to dig into that kind of now, but basically it tells us that, for example, if we're looking for like how we're dealing with missing values, is there noise and the data? You know it's a high cardinality, categorical variable, they're all different steps we would take so, for example, if it was a high cardinality, categorical variable that was originally a string right like, for example, I think, like maybe fi product class description. I remember one of the ones we looked at the other day. He had like first of all, was the type of vehicle and then a hyphen and then like the size of the vehicle. We might look at that and be like okay. Well, that was an important column.

Let's try like splitting it into two on and then take that bit, which is like a size of it and trying you know posit and convert convert it into an integer. You know we can try and do some feature engineering and basically until you know which ones are important, you don't know where to focus that feature engineering time. You can talk to your client, you know and say you know, or you know, and if you're doing this inside your workplace, you go and talk to the folks that, like we're responsible for creating this data, so in this, if you were actually working at a bulldozer Auction company - you might now go to the actual auctioneers and say I'm really surprised that couply system seems to be driving people's pricing decisions so much. Why do you think that might be? And they can say to you? Oh, it's actually, because only these classes of vehicles have capital systems, or only this manufacturer has couple of systems, and so frankly, this is actually not telling you about couple of systems, but about

something else. And oh hey that reminds me, that's that that's something else! We actually have measured that it's in this different CSV file I'll go, get it for you, but kind of helps. You focus your attention, so I hello little problem this weekend. As you know, I introduced a couple of crazy computations in into my random forest and all of a sudden they're like, oh my god. These are the most important variables ever squashing all of the others.

But then I got a terrible score and then is that, because now that I think I have my scores computed correctly, what I noticed is that the importance went through the roof, but the validation set was still bad or got worse. Is that, because, somehow, that computation allow the training to almost like an identifier map, exactly what the answer was going to be for training? But

5. 00:32:15

- **Two reasons why Validation Score is not good or getting worse: overfitting, and validation set is not a random sample (something peculiar in it, not in Train),**
- **The meaning of the five numbers results in 'print_score(m)', RMSE of Training & Validation, R^2 of Train & Valid & OOB.**
- **We care about the RMSE of Validation set.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Of course, that doesn't generalize to the validation set. Is that what is that? What I observed? Okay? So this there's two reasons why your validation score might not be very good um, let's go up here, okay, so we got these five numbers right. The rmse of the training, validation, r-squared of the training validation and the r-squared of the oeob okay. So there's two reasons and really in the end, what we care about like for this Kaggle competition is the rmse of the validation set. Assuming we've created a good validation set, so an Terrance's case he's saying this number. Is this thing I care about got worse when I did some feature engineering? Why is that? Okay, there's two possible reasons. Reason one is that you're overfitting if you're overfeeding, then your mobiie will also get worse if you're doing a huge data set with a small set RF sample. So you can't use a know. A B then instead create a second validation set, which is a random sample. Okay and and do that right so in other words, if your OB or your random sample validation set, is has got much worse, then you must be overfitting. I think in your case Terrence. It's unlikely: that's the problem, because random forests don't over fit that badly like it's very hard to get them to overfit that badly. Unless you use some really weird parameters, like only one estimator, for example, like once, we've got ten trees in there.

There should be enough variation that you're, you know you can definitely over fit, but not so much that you're going to destroy your validation score by adding a variable. So I'd think you'll find that's, probably not the case, but it's easy to check and if it's not the case, then you'll see that your oob score or your random sample validation score hasn't got worse, okay, so the second reason your validation score - can get worse. If your mobiie score hasn't got worse, you're, not overfitting, but your validation score is got worse. That means you're you're doing something that is true in the training set, but not true in the validation set. So this can only happen when your validation set is not a random sample. So, for example, in this bulldozes competition or in the grocery shopping competition, we've intentionally made a validation set. That's for a different date range it's for the most recent two weeks right, and so, if something different happened in the last two weeks to the previous weeks, then you could

totally break your validation set. So, for example, if there was some kind of unique identifier which is like different in the to date periods, then you could learn to identify things using that identifier in the training set, but then, like the last two weeks, may have a totally different set of ID's Or the different set of behavior could get a lot worse, yeah. What you're describing is not common though, and so I'm a bit skeptical.

It might be a bug, but

6. 00:35:50

- **How Feature Importance is normally done in Industry and Academics outside ML: they use Logistic Regression Coefficients, not Random Forests Feature/Variable Importance.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Hopefully, there's enough things you can now use to figure out if it is about, will be interested to hear what you learned. Okay, so that's that's feature importance, and so I'd like to compare that to how feature importance is normally done in industry and in academic communities outside of machine learning like in psychology and economics and so forth, and generally speaking, people in those kind of environments tend to Use some kind of linear regression, logistic regression general linear models, so they start with their data set and they basically say that was weird: oh okay, so they start with their data set and they say I'm going to assume that I know the kind of parametric relationship Between my independent variables and my dependent variable, so I'm going to assume that it's a linear relationship say or it's a linear relationship with a link function like a sigmoid to create logistic regression say, and so assuming that I already know that. I can now write this as an equation, so if I've got like x1 x2 so forth, right, I can say all right: my Y values are equal to ax 1 plus BX 2 equals y, and therefore I can find out the feature importance easily enough by just Looking at these coefficients and saying like which one's the highest, particularly if you've normalized the data first right so there's this kind of trope out there, it's it's very common, which is that, like this, is somehow more accurate or more pure or in some way better way Of doing feature importance, but that couldn't be further from the truth right, if you think about it, if you were like, if you were missing an interaction right or if you were missing a transformation you needed or if you have any way being anything less than a Hundred percent perfect in all of your pre-processing, so that your model is the absolute correct truth of this situation.

Right unless you've got all of that correct, then your coefficients are wrong right. Your coefficients are telling you in you're totally wrong model. This is how important those things are right, which is basically meaningless. So where else do the random forest feature importance? It's telling you in this extremely high parameter highly flexible, functional form with few. If any statistical assumptions. This is your future importance right. So I would be very cautious, you know, and and again I can't stress this enough when you, when you leave ence and when you leave this program, you are much more often going see. People talk about logistic regression coefficients then you're going to see them talk about random first variable importance, and every time you see that happen, you should be very, very, very skeptical of what you're seeing anytime, you read a paper in economics or in psychology or the marketing Department tells you that this regression or whatever every single time those coefficients are going to be massively biased by any issues in the model. Furthermore, if they've done so

much pre-processing that actually the model is pretty accurate, then now you're looking at coefficients that are going to be of like a coefficient of some principal component from a CA or a coefficient of some distance from some cluster or something at which Point they're very, very hard to interpret anyway they're not actual variables right, so they're kind of the two options I've seen when people try to use classic statistical techniques to do recover a variable importance, equivalent

7. 00:39:50

- **Doing One-hot encoding for categorical variables,**
- **Why and how works 'max_n_cat=7' based on Cardinality 49m15s, 'numericalize'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

I think things are starting to change slowly, you know there. There are some fields that are starting to realize that this is totally the wrong way to do things, but it's been, you know nearly 20 years since random forests appeared so it takes a long time. You know people say that the only way that knowledge really advances is when the previous generation dies and that's kind of true right. Like particularly academics, you know they make a career of being good at a particular sub thing, and you know often don't if you know it's not until the next generation comes along, that that people notice that oh that's actually longer no good way to do things, and I think that's what's happened here. Okay, so we've got now a model which isn't really any better as a predictive accuracy wise, but it's kind of we're getting a good sense that there seems to be like four main important things when it was made: the capital system, its size and its product classification. Okay, so that's cool. There is something else that we can do, however, which is we can do something called one hot encoding, so this is going to where we're talking about categorical variables, so remember a categorical variable. Let's say we had like a string high and remember the order we got was kind of back weird. It was high low medium, so it was in alphabetical order by default.

Right was their original category for like usage, banned or something, and so we mapped it to 0, 1, 2 right and so by the time it gets into our data frame. It's now a number, so the random forest doesn't know that it was originally a category. It's just a number right, so when the random forest is built, it basically says: oh, is it greater than 1 or not, or is it greater than naught or not? You know basically the two possible decisions it could have made for for something with like five or six bands. You know it could be that just one of the levels of a category is actually interesting right so like if it was like very high, very low or unknown right. Then we know that, like six levels - and maybe the only thing that mattered was whether it was like unknown - maybe like not nothing. It's sighs somehow impacts the price. And so, if we wanted to be able to recognize that, and particularly if like it just so happened that the way that the numbers were coded was it unknown ended up in the middle right, then what it's going to do is it's going to say. Okay, there is a difference between these two groups. You know less than or equal to two versus greater than two and then when it gets into this this leaf. Here it's going to say: oh there's a difference between these two between less than four and greater than or equal to four, and since going to take two splits to get to the point where we can see that it's actually unknown that matters.

So this is a little inefficient and we're kind of like wasting tree computation and like wasting tree computation, because every time we do a split we're having the amount of data, at least

that we have to do more analysis. So it's going to make our tree less rich, less effective. If we're not giving the data in a way that's kind of convenient for it to do the work it needs to do so. What we could do instead is create six columns. We could create a column, cord is very high, is very low, is high, is unknown, is low, is medium and h1 would be ones and zeros right, so the one or a zero. So we had six columns this one moment so having added six additional columns to our data set, the random first now has the ability to pick one of these and say like: oh: let's have a look at is unknown. There's one possible fit, I can do, which is one versus zero. Let's see if that's any good right, so it actually now has the ability, in a single step, to pull out a single category level, and so this, this kind of coding is called one-hot encoding and for many many types of machine learning model. This is like necessary. Something like this is necessary like if you are doing logistic regression. You can't possibly put in a categorical variable that goes not through five, because there's obviously no written linear relationship between that and anything right. So one part encoding a lot of people incorrectly, assume that all machine learning requires one pod encoding.

But in this case I'm going to show you how we could use it optionally and see whether it might improve things sometimes yeah, hi Jeremy. So we have six categories. Like in this case, would there would be any problems with adding a column for each of the categories? Oh Chris, in linear regression we saw we had to do it like. If there's six categories, we should only do it for five of them yeah so um it. You certainly can say: oh let's not worry about, adding is medium because we can infer it from the other. Five, I would say include it anyway, because like rather than otherwise the random forest would have to say is very high. Know is very low. Know is high, know, is unknown mode is low, know, okay and finally, on there right. So it's like five decisions to get to that point. So the reason in linear models that you need to not include one is because linear models hate collinearity, but we don't care about about that here. So we can do one hot encoding easily enough and the way we do it is we pass one extra parameter to proc DF, which is what's the max number of categories right. So if we say it's seven, then anything with less than seven levels is going to be turned into a one, hot encoded bunch of columns right. So in this case this has got six levels, so this would be one hot encoded where else like zip code has more than six levels, and so that would be left as a number and so, generally speaking, you obviously probably wouldn't want a one hot in code.

Zip code right because that's just going to create masses of data memory, problems, computation problems and so forth right, so so this is like another parameter that you can play around with. So if I do that, try it out run the random forest. As per usual, you can see what happens to the r-squared of the validation set and to the rmse of the validation set, and in this case I found it got a little bit worse. This isn't always the case and it's going to depend on your data set. You know: do you have a data set where you know single categories tend to be quite important or not in this particular case, it didn't make it more predictive. How what it did do is that we now have different features right, so proc TF puts the name of the variable and then an underscore, and then the level name, and so interestingly, it turns out that, where else before it said that enclosure was somewhat important when We do it as one hot encoded, it actually says. Enclosure erupts with AC is the most important thing so for at least the purpose of like interpreting your model, you should always try one hot encoding. You know quite a few of your variables, and so I often find somewhere around six or seven is pretty good. You can try like making that number as high as you can, so that it doesn't take forever to compute and the feature importance doesn't include like really tiny levels that aren't interesting, so that's kind of up to you to play it play around with, but in this Case like this is actually I found this very interesting.

It clearly tells me I need to find out what enclosure erupts with AC is. Why is it important because, like means nothing to me right and but it's the most important thing, so I should go figure that out so then I had a question: you plus it. So can you explain how changing the max number of categories worse? Because for me, it just seems like there's five categories or site categories: oh yeah, sorry! So it's it's just like all it's doing is saying like okay, here's, a column called zip code, here's a column, called usage band and here's a column sex right. I don't know whatever right and so, like zip code has, whatever five thousand levels the number of levels in a category we call its cardinality okay, so it has a cardinality of five thousand usage banned. Maybe has a cardinality of six sex has maybe a cardinality of so when proc TF goes through and it says okay, this is a categorical variable. Should i one-hot encode it? It checks the cardinality against max and hats and says all five thousand is bigger than seven. So I don't one hot encoder and then it goes to usage. Band 6 is less than 7. I do one hot encode it goes to. Sex 2 is less than 7. I do want to encode it, so it just says for each variable. How do I decide whether the one hot encoded or not? We are keeping legal in cause? No, once we decide to one hot in code, it does not keep the original variable. Maybe the best will be an interval. Well, you don't need a labeling code if the.

If so, if the best is an interval, it can approximate that with multiple one hot encoding levels. Yeah. So like you know it's a the. The truth is that each column is going to have some. You know different. You know, should it be label encoded or not, you know which you could make on a case-by-case basis. I find in practice it's just not that sensitive to this, and so I find like just using a single number for the whole data set. Gives me what I need, but you know if you were building a model that really had to be as awesome as possible and you had lots and lots of time to do it. You can go through men, you know, don't use property if you can go through manually and decide which things to use dummies or not your you'll see in the code. If you look at the code for property, F, Rock D F right like I - never want you to feel like the code that happens to be in the fastai library, is the code that you're limited to right. So where is that done? You can see that the max n cat gets passed to numerical eyes and numerical eyes simply checks. Okay, is that a numeric type and it's the number of categories either not in pass to us at all or we've got more unique values than there are categories and if so, we're going to use the categorical codes. So for any column, where that's where it's skipped over that right, so it's remained as a category then at the very end we just go. Pandas get dummies, we pass in the whole data frame and so a pandas get.

That means you pass in a whole data frame, it checks for anything, that's still a categorical variable and it turns it into a dummy variable which is another way of saying a one-pot encoding. So you know with that kind of approach you can easily override it into your own dummy verification. Variable ization did you have a question, so some data has a quite obvious order like if you have like a grading system like food, bad or whatever things like that. There's an order to that and showing that order by doing the dummy variable thing, probably will your benefit, so is there a way to just force it to leave alone, one variable just like invert, it or yourself, not not in the library, and to remind you like Unless we explicitly do something about it, we're not going to get that order. So when we, when we import the data, so this is in Lesson one RF: we showed how, by default, the categories are ordered alphabetically and we have the ability to order them properly. So yeah, if you've actually made an effort to turn your ordinal variables into proper ordinals using prop D F, can destroy that if you have max MCATs, so the simple thing, the simple way to avoid that is, if we know that we always want to use the Codes for usage banned rather than the you know like never one hot encoder. You could just go ahead and replace it right.

You could just say: okay, let's just go D, F dot; u s! -- taband equals DF q, suspend cat codes and it's now an integer and so it'll never get page all right. So we kind of already seen how variables, which are basically measuring the same thing, can kind of confuse our

8. 00:55:05

- **Removing redundant features using a dendogram and '.spearmanr()'for rank correlation, 'get_oob(df)', 'to_drop = []' variables, 'reset_rf_samples()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Variable importance and there can also make our random forest slightly less good, because it requires like more computation to do the same thing. There's more columns to check so I'm going to do some more work to try and remove redundant features, and the way I do that is to do something called a dendrogram and it's a kind of hierarchical clustering. So cluster analysis is something where you're trying to look at objects. They can be either rows in the data set or columns and find which ones are similar to each other. So often you'll see people particularly talking about cluster analysis. They normally refer to rows of data and they'll say like oh, let's plot it right and like oh there's a cluster and there's a cluster right, a common type of cluster analysis time permitting we may get around to talking about this in some detail, is called k-means, Which is basically where you assume that you don't have any labels at all and you take basically a couple of data points at random and you gradually find the ones that are near to it and move them closer and closer to centroids. And you kind of repeat it again and again, and it's an iterative approach that you basically tell how many clusters you want and it'll tell you where it thinks that classes are. I really I don't know why, but I really under use technique. 20. 30 years ago.

It was much more popular than it is today is hierarchical, clustering, hierarchical also known as agglomerated clustering and in hierarchical order, agglomerative clustering. We basically look at every pair of option up every pair of objects and say: okay, which two objects are the closest alright. So in this case, we might go okay. Those two objects are the closest and so we've kind of like delete them and replace it with the midpoint of the two and then okay here, the next two closest if we delete them and replace them with the midpoint of the two and you keep doing that Again again right since we're kind of removing points and replacing them with their averages, you're gradually reducing a number of points by pairwise combining and the cool thing is. You can plot that, like so right? So if, rather than looking at points, you look at variables, we can say okay, which two variables are the most similar. It says: okay, say year and sale elapsed, they're very similar. So the kind of horizontal axis here is how similar are the two points that are being compared right. So, if they're closer to the right, that means they're very similar, so sale year and sale elapsed have been combined and they were very similar again, it's like okay, as you know, it'll be like correlation coefficient or something like that.

You know in this particular case what I actually did, so you get to tell it so in this case I actually used Spearman's R, so you guys familiar with correlation coefficients already all right, so correlation is cut as almost exactly the same as the r-squared right, but It's between two variables, rather than a variable, and it's prediction. The problem with a normal correlation is that if the I create a new workbook here, if you have data that looks like this, then you can do a correlation and you'll get a good result right. But if you've got data which

looks like this right and you try and do a correlation, it assumes linearity. That's not very good right! So there's a thing called a rank correlation, a really simple idea: it's replace every point by its rank right, so, instead of like so, we basically say: okay, this is the smallest, so we'll call that one there's the next one three is next one, four five right. So you just replace every number by its rank and then you do the same for the y-axis so that 1, 2, 3 and so forth. Right and so then you do it like a new plot, where you don't plot the data, but you plot the rank of the data and, if you think about it, the rank of this data set is going to look an exact line, because every time something was Greater on the x-axis, it was also greater on the y-axis. So if we do a correlation on the rank, that's called a rank correlation, okay, and so because I want to find the columns that are similar in a way that the random forest would find them similar.

Random forests, don't care about linearity, they just care about ordering. So a rank correlation is the the right way to think about that. So Spearman's are is, is the name of the most common rank correlation, but you can literally replace the data with its rank and chuck it at the regular correlation and you'll get basically the same answer. The only difference is in how ties are handled. It's a pretty minor issue if you had like a full parabola in that rank, correlation you'll will not write right. It has to be has to be monotonic, yeah, yeah, okay, so once I've got a correlation matrix, there's, basically a couple of standard steps. You do to turn that into a dendogram, which I have to look up on stackoverflow each time I do it, you basically turn it into a distance matrix, and then you create something that tells you you know which things are connected to which other things hierarchically. So this kind of these two and this step here, like just three standard steps that you always have to do to create a dendogram, and so then you can plot it, and so alright so say your and sell a lot soon to be measuring. Basically, the same thing, at least in terms of rank, which is not surprising because they elapsed is the number of days since the first day in my data set. So obviously, these two are nearly entirely correlated with some ties, browser tracks and hydraulics flow and coupla system.

All seem to be measuring the same thing, and this is interesting because remember coupla system it said was super important right, and so this rather supports our hypothesis there's nothing to do with whether it's a coupler system, but whether it's whatever kind of vehicle it is. It has these kind of features. Product group and product groups desk seem to be measuring the same thing. If I base model on fi model desk seem to be measuring the same thing, and so once we get past that everything else like suddenly, the things are further away. So I'm probably going to not worry about those. So we're going to look into these one. Two. Three four groups that are very similar: could you pass that over there, the citizen that grabbed that the similarity between stick, glint and enclosure is higher than with stick lens and anything, that's higher yeah? Pretty much I mean it, it's a little hard to interpret, but given that stick length and enclosure don't join up until way over here yeah, it would strongly suggest that then, that they're a long way away from each other. Otherwise you would expect them. We were joined up earlier. I mean it's it's possible to construct like a synthetic data set where you kind of end up joining things that were close to each other through different paths. So you've got to be a bit careful, but I think it's fair to is probably assume that stick length or enclosure are probably very different, so they are very different, but would they be more similar than, for example, stick length and sale day of year? No, which is in a very top, no there's nothing to suggest that here because, like the key point, is to notice where they sit in this tree right and they both that they sit in totally different halves of the tree.

Thank you, but really to actually know that the best way would be to actually look at this p.m. and our correlation matrix, and if you just want to know how similar is this thing, or this thing the Spearman, our correlation matrix, tells you that. Can you plus that over there, so

today's we are passing the leader cream right? Second, we are passing the cream. This is just a data frame, so we're passing in DF Cape. So that's the data frame containing the whatever it was 30 or so features that our random forest thought was interesting. So there's no random first being used here, the measure the distance measure is being done entirely on rent correlation. So what I then do is I take these these groups right and I create a little function that I call get out of fans score right, which is it does a random forest for some data frame. I make sure that I've taken that data frame and split it into a training and validation set, and then I call fit and return the oeob score right. So, basically, what I'm going to do is I'm going to try removing each one of these one. Two three. Four: five: six, seven and eight nine or so variables, one at a time and see which ones I can remove and it doesn't make the oob score get worse and each time I run this, I get slightly different results. So actually it looks like last time I had seven things, not not eight things, so you can see. I just do a loop through each of the things that I'm thinking like.

Maybe I could get rid of this because it's redundant and I print out the column name and the oeob score of a model that is trained after dropping that one column, okay, so the oeob score on my whole data frame is point eight, nine and then, after Dropping each one of these things they're, basically none of them get much worse. Sale elapsed is getting quite a bit worse than say all year, but, like it looks like pretty much everything else I can drop with, like only like a third decimal place problem. So obviously, though, you've got to remember the dendogram, let's take fi model discs and Fi based model right, they're, very similar to each other right. So what this says isn't that I can get rid of both of them right. I can get rid of one of them because they're basically measuring the same thing. Okay. So so then I try it. I say: okay, let's try getting rid of one from each group, say: oh yeah, F by based model and grouse attracts okay and like let's now have a look. It's like okay, I've gone from point. Eight. Nine! Oh two point: eight, eight, eight, it's like again so close as to be meaningless, so that sounds good. Simpler is better. So I'm now going to drop those columns from my data frame and then I can try running the full model again and I can see you know so reset our air samples means I'm using my whole data frame.

My whole bootstrap sample used for tea estimators and I've got 0.90 seven okay, so I've now got a model which is smaller and simpler and I'm getting a good score.

9. 01:07:15

- **Partial dependence: how important features relate to the dependent variable, 'ggplot() + stat_smooth()', 'plot_pdp()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

For so at this point, I've now got rid of as many columns as I feel I comfortably can ones that either didn't have a good feature importance or were highly related to other variables, and the model didn't get worse significantly whenever when I removed them. So now, I'm at the point where I want to try and really understand my data better by taking advantage of the model and we're going to use something called partial dependence. And again, this is something that you could like using the Carroll kernel and lots of people are going to appreciate this, because almost nobody knows about partial dependence and it's a very, very powerful technique. What we're going to do is we're going to find out for the features that are important. How do they relate to the dependent variable right? So let's have a look right. So, let's again since

we're doing interpretation, we'll set set our samples to 50,000 to run things quickly, we'll take our data frame. We'll get our feature importance and notice that we're using Max and Kat, because I'm actually pretty interested in terms of interpretation and seeing the individual levels, and so here's the top 10. And so let's try and learn more about those top 10. So year made is the second most important. So one obvious thing we could do would be to plot year made against sale elapsed because, as we've talked about already like it just seems to make sense that both important, but it seems very likely that they kind of combine together to find like how old was The product when it was sold, so we could try plotting year, made against sale elapsed to see how they relate to each other.

And when we do, we get this very ugly graph, and it shows us that year made actually has a whole bunch. That are a thousand right. So clearly, you know this is where I would tend to go back to the client or whatever and say: ok, I'm guessing that these bulldozers weren't actually made in the Year 1000 and they would presumably say to me: oh yes, they're ones where we don't know where It was made, you know, maybe before 1986 we didn't track that or maybe the things that are sold in Illinois. We don't have that data provided or or whatever they tell us some reason. So in order to understand this plot better, I'm just going to remove them from this interpretation section of the analysis. So I'm just going to say: ok, let's just grab things where year made is greater than 1930 ok. So, let's now look at the relationship between year made and sale, press and there's a really great package called GG plot. Gg plot originally was an package G G stands for the grammar of graphics, and the grammar of graphics is like this very powerful way of thinking about how to produce charts in a very flexible way. I'm not going to be talking about it much in this class. There's lots of information available online, but I definitely recommend it as a great package to use ggplot, which you can pip install. It's part of the fastai environment. Already ggplot in python has basically the same parameters and API as the R version.

The R version is much better documented, so you should read it's documentation to learn how to use it, but basically you say: okay, I want to create a plot of this data frame. Now, when you create plots most of the datasets you're using, are going to be too big to plot, as in like, if you do a scatter plot, it'll create so many dots that it's just a big mess. It'll take forever and remember when you're plotting things you just you're you're looking at it right, so there's no point putting something with a hundred million samples when, if you're only used a hundred thousand samples, it's going to be pixel identical right. So that's why I call get sample first, so get sample just grabs a random sample. Okay, so I'm just going to grab five hundred points for now. Okay, so I've got a grab. Five. At a point from my data frame, I got a plot a year made against sale price. A EES stands for aesthetic. This is the basic way that you set up your columns in ggplot, okay, so this says to plot these columns from this data frame and then there's this weird thing and GG plot. Where plus means basically add chart elements? Okay, so I'm going to add a smoother. So most of the very very often you'll find that a scatter plot is very hard to see. What's going on because there's too much randomness or else a smoother basically creates a little linear regression for every little subset of the graph, and so it kind of joins it up and allows you to see a nice smooth curve.

Okay. So this is like the main way that I tend to look at univariate relationships and by adding standard error equals true. It also shows me the confidence interval of this smoother right. So low S stands for locally weighted regression, which is this idea of like doing kind of out doing lots of little mini regressions. So we can see here. The relationship between year made and sale price is kind of all over the place right, which is like not really what I would expect. I would. I would have expected that more recent stuff it sold more recently would probably be

like more expensive because of inflation, and because there like more current models and so forth, and the problem is that when you look at a univariate relationship like this there's a whole lot Of collinearity, going on a whole lot of interactions that are being lost, so, for example, why did the price drop yeah? Is it actually because, like things made between 1991 and 1997, are less valuable or is actually because most of them were also sold during that time? And actually there was like maybe a recession then or maybe it was like products sold during that time - a lot more people but buying types of vehicle that were less expensive, like there's all kinds of reasons for that, and so again as data scientists, one of the Things are going to keep seeing is that at the companies that you join, people will come to you with with these kind of univariate charts where they'll say like.

Oh, my god, our sales in Chicago have disappeared. That got really bad or people aren't clicking. On. This add anymore and they'll. Show you a chart that looks like this and they'll be like what happened and most of the time you'll find the answer to the question. What happened is that there's something else going on right, so I actually are in Chicago last week. Actually, we were doing a new promotion and that's why you know revenue went down it's not because people are buying stuff in Chicago anymore. It's because the prices were lower, for instance. So what we really want to be able to do is say: well, what's the relationship between sale, price and year made all other things being equal, so all other things being equal basically means if we sold something in 1990 versus 1980, and it was exactly the same Thing exactly the same person in exactly the same option so on and so forth. What would have been the difference in price, and so to do that we do something called a partial dependence plot, and this is a partial dependence plot. There's a really nice library which nobody's heard of called PDP, which does these partial dependence plots and what happens is this we've got our sample of 500 data points right and we're going to do something really interesting we're going to take each one of those hundred randomly Chosen options and we're going to make a little data set out of it right so, like here's, our here's elf come on here's our data set of like 500 options and here's our columns, one of which is the thing that we're interested in which is year made.

So here's year made okay and what we're going to do is we're now going to try and create a chart where we're going to try and say all other things being equal in 1960? How much did bulldozers cost? How much did things cost in options, and so the way we're going to do that is we're going to replace the year made column with 1960 we're going to copy in the value 1960 again and again and again, all the way down right. So now every row the year made is 1960 and all of the other data is going to be exactly the same and we're going to take our random forest. We're going to pass all this through our random forest to predict the sale price. So that will tell us for everything that was auctioned. How much do we think it would have been sold for if that thing was made in 1960, and that's what we're going to plot here all right? That's the price we're going to put here and then we're going to do the same thing for 1961. Alright, we're going to replace all these and do 1961 yeah, so to be clear: we've already fit the random forest, yes and then we're just passing a new year and seeing what it determines the price should be yeah. So this is a lot like the way we did feature importance, but rather than randomly shuffling the column, we're going to replace the column with a constant value.

All right, so randomly shuffle in the column, tells us how accurate it is when you don't use that column anymore, replacing the whole column with a constant tells us or estimates for us how much we would have sold that product for in that auction on that day. In that place, if that product had been made in 1961 right, so we basically then take the average of all of the sale prices that we calculate from that random first, and so we drew it in 1961 and we get this

value right. So what the partial dependence plot here shows us is each of these light. Blue lines actually is showing us all 500 lions. So it says for row number 1 in our data set if we sold it in 1960, we're going to index that to 0 right so call that zero right if we sold it in 1970, that particular auction would have been here if we sold it in 1980. I would have been here if he sold in 1990 would have been here, so we actually plot all 500 predictions of how much every one of those 500 auctions would have gone for if we replace it before replacing a year made with each of these different values And then then, this dark line here is the average right. So this tells us how much would we have sold on average all of those options for if all of those products were actually made in 1985, 1990, 1993, 1994 and so forth, and so you can see.

What's happened here is at least in the period where we have a reasonable out of data, which is since 1990. This is basically a totally straight line, which is what you would expect right, because if it was sold on the same date and it was the same kind of tractor, it sold to the same person in the same option house, then you would expect more recent vehicles To be more expensive because of inflation, and because they're they're newer right, they're, not they're, not as secondhand, and you would expect that relationship to be roughly linear and that's exactly what we're finding ok. So by removing all of these externalities, it often allows us to see the truth much more clearly as a question the back. Can you pass that back there you're done. Ok, so um this, this partial dependents plot concept is something which is using a random forest to get us a more clear interpretation of what's going on in our data, and so the steps were to first of all, look at the feature importance to tell us like Which things do we think we care about and then to use the partial dependence plot to tell us what's going on on average right there's another cool thing we can do with PDP as we can use clusters and what clusters does is it uses cluster analysis? To look at all of these each one of the 500 rows and say to some of those 500 roads kind of move. In the same way and like we could kind of see, it seems like there's a whole lot of rows, that kind of go down and then up and there seems to be a bunch of rows that kind of go up and then go flat like it does Seem like there's some kind of different types of behaviors being hidden, and so here is the result of doing that.

Cluster analysis right is, we still get the same average, but it says here kind of the five most common shapes that we see, and this is where you could then go in and say all right. It looks like some kinds of vehicle. Actually, after 1990, their prices are pretty flat and before that they were pretty linear some kinds of vehicle and of exactly the opposite, and so like different kinds of vehicle. Have these different shapes right, and so this is something you could dig into. I think it was one at the back. Oh you could, okay. So what we're going to do with this information? Well, the purpose of interpretation is to learn about a data set, and so why do you want

10. 01:21:50

- **What is the purpose of interpretation, what to do with that information ?**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

To learn about a data set, it's because you it's, because you want to do something with it right. So in this case, it's not so much something. If you're trying to win a cogwheel competition, I mean it can be a little bit like some of these insights. Might make you realize all I could transform this variable or create this interaction or whatever obviously feature importance is

super important for cowgirl competitions, but this one much more for like real life. You know so this is when you're talking to somebody and you say to them like okay, those plots you've been showing me which actually say that, like there was this kind of dip in prices, you know based on, like things made between 1990 and 1997, there wasn't Really you know, actually it was they were increasing. There was actually something else going on at that time. No, it's basically the thing that allows you to say like so. Whatever this outcome, I'm trying to drive in my business is: this is how something's driving it all right. So if it's like I'm looking at, you know kind of advertising technology. What's driving clicks that I'm actually digging into say? Okay, this is actually how clicks are being driven. This is actually the variable, that's driving it. This is how it's related. So, therefore, we should change our behavior in this way. That's really the goal of any model.

I guess there's two possible goals: 1 goal of a model is just to get the predictions like if you're doing hedge fund trading, you probably want to know what the price of that equity is going to be if you're doing insurance. You probably just want to know how much claims that guy's going to have, but probably most of the time, you're actually trying to change something about how you do business, how you do marketing how you do just sticks, so the thing you actually care about is how The things are related to each other. All right, I'm sorry. Can you explain again when you scroll up and you were looking at the sale, pricier may looking at the entire model, and you saw that dip and you said something about that dip didn't signify what we thought it did. Can you explain why yeah? So this is like a classic boring, univariate plot right, so this is basically just taking all of the dots all of the options plotting year made against sale, price and we're gon na just fitting a rough average through them, and so true that products made between 1992 And 1997 on average in our data set, are being sold for less so, like very often in business. You'll hear somebody look at something like this and they'll be like. Oh, we should. We should stop auctioning equipment that is made in that year in those years because, like we're getting less money, for example, but if the truth actually is that during those years, it's just that people were making more small industrial equipment where you would expect it to be Sold for less and actually our profit on, it is just as high, for instance, or during those years.

It's not that it's not things made during those years now would have repeat cheaper it's that during those years when we were selling things in those years, they were cheaper because, like there was a recession going on. So if you're, trying to like, actually take some action based on this, you probably don't just care about the fact that things made in those years are cheaper on average. But how does that impact today? You know so so this this approach, where we actually say, let's try and remove all of these externalities. So if something is sold on the same day to the same person of the same kind of vehicle, then actually have, as year made impact price, and so this basically says, for example, if I am deciding what to buy at an option, then this is kind of Saying to me, okay, like getting a more recent vehicle on average, really does on average, give you more money, which is not what the kind of the naive univariate plot said, that, because it's Tyler, for like this bulldozer bulldozers made in 2010, probably are Not close to the type of bulldozers that were made in 1960 right and, if you're taking something that would be so very different, like a 2010 bulldozer and then trying to just drop it to say. Oh, if it was made in 1960 that may cause poor prediction at a point, because it's so you're outside absolutely rainy. Absolutely so you know, I think, that's a good point. It's you know it's a limitation, however.

Random forest is if you're got a kind of data. Point that's like over client, you know which is kind of like in a part of the space that it's not seen before, like maybe people didn't put air

conditioning really in bulldozers in 1960 and you're saying how much would this bulldoze over their conditioning have gone for 1960, you don't really have any information to know that. So you know you it's a it's it's. This is still the best technique I know of, but it's it's not perfect, and you know you kind of hope that the trees are still going to find some useful truth. Even if, though, it hasn't seen that combination of features before but yeah, it's something to be aware of so you can also do the same thing in a PDP interaction plot and a PDP interaction plot, which is really what I'm trying to get to here is like How to sail elapsed and year made together impact price, and so, if I do a PDP interaction plot, it shows me sail elapsed versus price. It shows me year made versus price and it shows me the combination versus price remember. This is always log of price. That's why these prices look weird right, and so you can see that the combination of sale elapsed and year made is, as you would expect later dates. So more or less time is giving me I'm sorry, it's the other way around. Isn't it so the higher crisis? Those where there's the least elapsed and the most recent year made so you can see here, there's the univariate relationship between sale, elapsed and price, and here is the univariate relationship between year made and price.

And then here is the combination of the two. It's enough to see like clearly that these two things are driving christs together. You can also see these are not like simple diagonal lines, so it's kind of some interesting interaction going on and so based on. Looking at these plots, it's enough to make me think. Oh, we should maybe put in some kind of interaction term and see what happens. So, let's come back to that in a moment, but let's just look at a couple more remember in this case I did one hot encoding way back at the top here I said max and cat equals seven, so I've got like enclosure erupts with AC. So if you've got one hot encoded variables, you can pass an array of them to pit plot PDP and it'll treat them as a category right, and so in this case, I'm going to create a PDP plot of these three categories. I'm going to call it enclosure, and I can see here that enclosure erupts with AC on average are more expensive than enclosure erupts and enclosure arose. It actually looks like enclosure erupts from closure, erupts are pretty similar, or else erupts with AC is higher. So this is, you know at this point you know I probably being fine to hop into Google and like type erupts and erupts and find

11. 01:30:15

- **What is EROPS / OROPS ?**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Out what the hell these things are, and here we go so it turns out that erupts is enclosed rollover, protective structure, and so it turns out that if your your bulldozer is fully enclosed, then optionally, you can also get air conditioning. So it turns out that actually this thing is telling us whether it's got air conditioning. If it's an open structure, then obviously you don't have air conditioning at all. So that's what these three levels are, and so we've now learnt all other things being equal. The same bulldozer sold at the same time, built at the same time, sold to the same person is going to be quite a bit more expensive is if it has air conditioning than if it doesn't ok. So again, we're kind of getting this nice interpretation ability - and you know now that I spent some time with this data set. I'd certainly noticed that this you know knowing this is the most important thing you do notice that there's a lot more air conditioned bulldozers nowadays and they used to be, and so there's definitely an interaction between kind of date and that so based on the earlier interaction Analysis I've tried first of all setting

everything before 1950 to 1950s. It seems to be some kind of missing value. I've been set age to be equal to sale year year made, and so then I try running a random forest on that, and indeed page is now. The single biggest thing sale elapsed is way back down here year.

Made is back down here, so we've kind of used this to find an interaction, but remember, of course, a random forest can create, or it can create an interaction through having multiple split points. So we shouldn't assume that this is actually going to be a better result and in practice I actually found when I looked at my score and my rmse, adding age was actually a little worse and we'll see about that.

12. 01:32:25

- **Tree interpreter**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Later, probably in the next lesson, ok, so one last thing is tree interpreter, so this is also in the category of things that most people don't know exists, but it's super important, almost pointless for like cattle competitions but super important for real life and here's the idea. Let's say you're an insurance company and somebody rings up and you give them a quote and they say: oh that's, five hundred dollars more than last year. Why? Okay, so in general, you've made a prediction from some model and somebody asks why, and so. This is where we use this method called tree interpreter and what tree interpreter does is it allows us to take a particular row so in this case we're going to pick row number zero right. So here here is row: zero right, uh. Presumably this is like year made, I don't know what all the codes stand for, but like his is all of the columns in row 0. What I can do with a tree interpreter is, I can go t i dot predict pass in my random forest pass in my row, so this would be like this particular customers, insurance information or this in this case this particular option right and it'll. Give me back three things: the first is the prediction from the random forest. The second is the bias. The bias is basically the average sale price across the whole original data set right so, like remember, you know random forest.

We started with single trees. Oh, we haven't got to draw in there anymore, but remember we started with a single tree in our random forest and we split it once and then we spit that once and then we split that one straight. We said like oh: what's the average value for the whole data set, then what's the average value for those where the first split was true and then what's the average value where the next that was also true until eventually you get down to the leaf nodes where You've got the average value you predict right, so you can kind of think of it. This way, if this for a single tree, if this is our final leaf, node right - maybe we're predicting like nine point, one right and then maybe the average log sale price for the whole. The whole lot is like ten point: two right: that's the average through all the options, and so you could kind of like work your way down here. So let's go and create this. That's actually go and run this, so I can see it okay. So let's go back and redraw this single tree you'll find like in Jupiter notebooks, often a lot of the things we create like videos, progress bars and stuff. They don't know how to like save themselves to the file. So you'll see just like a little string here, and so you actually have to rerun it to create the string. So this was the single tree that we created so the whole dataset had an average log sale price of 10.2. The data set for those with capital system equals true had an average of ten point.

Three, the data set for capital system equals true enclosure. Less than point lesson two was nine point: nine and then eventually we get all the way up here and also a model ID less than forty five. Seventy three, it's ten point two, so you could kind of like say, okay. Why did this particular row? Let's say we had a row that ended up over in this leaf node. Why did we predict him point two? Well, it's because we start with ten point, one nine and then because the capitalist system was was was less than point five, so it was actually false. We added about point two to that, so we went from ten point one to ten point three right. So ten point two to ten point three. So we added a little bit because if this one is true and then to go from ten point three to nine point: nine so because enclosure is less than two we subtracted about 0.4 and then because model ID was less than 45-hundred. We added about point seven right, so you can see like with a single tree. You could like break down like. Why is it that we predicted ten point two retinas like and each one of these decision points we're adding or subtracting a little bit from the value. So what we could then do is we could do that for all the treats and then we could take the average. So every time we see enclosure did we increase or decrease the value and how much, by every time we see model ID, did we increase? What decrease the value and how much by, and so we could take the average of all of those and that's what ends up in this thing called contributions, so here is all of our predictors, and here is the value of each, and so this is telling us And I've sorted them here that the fact that this thing was made in 1999 was the thing that most negatively impacted our prediction and the fact that the age of the vehicle was 11 years was more most positively impacted um.

I think you actually needs a sort. After you zip them together, they seem to be sort of negative point. Five. Well, my bunions are sorted, but then they're just reassigned to the columns in the original order, which is what's. Thank you thank you. That makes perfect sense. Yes, we need to do an index sort. Okay, thank you. We will make sure we fix that by next week, so we need to sort columns by the index from contributions. So then there's this thing called bias, and so the bias is just the average, but before we start doing any splits right. So if you basically start with the average log of value and then we went down each tree and each time we saw a year made, we had some impact couple systems, some impact product, size, some impact and so forth, right, . Okay. So I think what we might do is we might come back to because we could have out of time. We might come back to tree interpreter next time, but the basic idea. This is the last. This is the last of our key interpretation points, and the basic idea is that we want some ability to not only tell us about the model as a whole and how it works on average. But to look at how the model makes predictions for an individual row and that's what we're doing here: okay, great next, everybody see you on this way.

Lesson 05

Outline

- Summary of Random Forests
- Data needs to be numeric
- Categories go to numbers
- Subsampling in different trees
- Tree size
- Records per node
- Information Gain (improvement)
- Repeat process for different subsetes
- Each tree should be better
- Trees should not be correlated
- Min Leaf Samples
- Max Features
- n_jobs
- oob
- interpretting OOB vs. Training vs. Test score
- Feature Importance Deep dive
- One hot encoding
- Redundant features
- Partial Dependence

Video Timelines and Transcript

1. 00:00:04

- **Review of Training, Test set and OOB score, intro to Cross-Validation (CV),**
- **In Machine Learning, we care about Generalization Accuracy/Error.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, so welcome back so we're going to start by doing some review and we're going to talk about test sets. Training sets, validation, sets and oob. Something we haven't covered yet, but we will cover in more detail. Later is also cross validation. But I'm going to talk about that as well right, so we have a data set with a bunch of rows in it and we've got some dependent variable, and so what's the difference between my machine learning and kind of pretty much any other kind of work that The the difference is that in machine learning, the thing we care about is the generalization accuracy or the generalization error. Where else in pretty much everything else. All we care about is is how well we could have mapped to the observations, all stop, and so this this thing about generalization is the key, unique piece of with machine learning, and so, if we want to know whether we could doing a good job of machine Learning we need to do know whether we're doing a good job of generalizing. If we don't

know that we know nothing right by generalizing, do you mean like scaling being able to scale larger? No III, don't mean scaling at all. So scaling is an important thing in many many areas. It's like okay, we've got something that works. I'm on my computer with ten thousand items. I do need to work, make it work on ten thousand items per second or something so scaling is important, but not just a machine learning for just about everything we put in production.

Generalization is where I say: okay, here is a model that can predict cats from dogs. I've looked at five pictures of cats, five pictures of dogs and I've built a model that is perfect and then I look at a different set of five cats and dogs, and it gets them all wrong. So, in that case, when it learned, was not a good Street, a cat and a dog that it learned what those five exact cats looked like in those five exact dogs, the play - or I built a model of predicting grocery sales for a particular product. So for toilet rolls in New Jersey last month and then I go and put it into production and it scales great. In other words, it can have the great latency. I don't have a high CPU load, but it fails to predict anything well other than toilet rolls in New Jersey. It also it turns out. It only did it well for last month, not the next month, so these are all generalization failures, so the most common way that people check for the ability to generalize is to create a random sample. So they'll grab a few rows at random and pull it out into a test set and then they'll build all of their models on the rest of the rows and then, when they're finished they'll check that the accuracy they got on there. So the rest of the rows are called the training set everything else, everything else we could call the training set, and so, at the end of their modeling process, on the training set, they got an accuracy of 99 percent of predicting cats from dogs. At the very end, they check it against a test set to make sure that the model really does generalize.

Now the problem is what, if it doesn't right so, okay? Well, I could go back and change. Some hyper parameters. Do some data augmentation and whatever else, try to create a more generalizable model and then I'll go back again after doing all that and check, and it's still no good and I'll keep doing this again and again until eventually, after fifty attempts, it does generalize. But does it really generalize, because maybe all I've done is accidentally found this one which happens to work just for that test set because I've tried 50 different things right, and so, if I've got something which is like right? Coincidentally, 0.05, 5 percent of the time. They're. Not very likely to accidentally get a good result, so what we generally do is we put aside a second data, set they'll, get a couple more of these and put these aside into a validation set. It's an audacious set right and then everything - that's not in the validation or tests is now training. And so what we do is we train a model check it against the validation to see if it generalizes do that a few times, and then, when we finally got something we were like okay, we think this generalizes successfully based on the validation set and then at The end of the project, we check it against the test set yeah. So, basically, if I making this two layer test, that validation said if he gets one right, the other one wrong, you're kind of double-checking, your errors, it's checking that we have an overfit to the validation set.

So if we're using the validation set again and again, then we could end up not coming up with a generalizable sort of hyper parameters and a set of private creditors that just so happened to work on the training set and the validation set. So so, if we try 50 different models against the validation set and then at the end of all that, we then check that against the test set and it's still generalized as well, then we're kind of going to say. Okay, that's good. We've actually come up with generalizable model. If it doesn't, then that's going to say: okay, we've actually now overfit to the validation set at which point you're kind of in trouble right because you don't you know, you don't have anything left behind right. So the idea is to use effective techniques

during the modeling so that so that doesn't happen right. But but if it's going to happen, you want to find out about it like you need that test set to be there, because otherwise, when you put it in production and then it turns out that it doesn't generalize, that would be a really bad outcome right. You end up with less people clicking on your ads or selling less with your products or providing car insurance to very risky vehicles or whatever so just make sure to need to ever check if the validation set and the test antics is coherent or you just keep Tested so if you've done what I've just done here, which is to randomly sample, there's no particular reason to check as long as there as long as they're big enough right, but we're going to come back to your question in a different context.

In just a moment. Now another trick: we've learned for renin forests is a way of not needing a validation set and the way what we learnt was to use instead use the oob era for the OO be scored, and so this idea was to say well, every time we train a Tree in a random forest, there's a bunch of observations that are held out anyway, because that's how we get some of the randomness. And so let's calculate our score for each tree, based on those held out samples and therefore the forest. By averaging the trees that that each row was not part of training, okay and so the oob score gives us something which is pretty similar to the validation score, but on average it's a little less good. Can anybody either remember or figure out why, on average, it's a little less good, quite a subtle one, don't give it to Kenji, I'm not sure, but is it because you are treating like you're doing every kind of probe pre-processing on your tests and so the OB Score is reflecting the performance on testing set no for the other piece, because not using the test set at all the other Peace Corps is using the held out rows in the training set at page tree. So I mean Z. You are basically testing each tree of some data from Z. Training set. Yes, so you are. You have the potential of over feeding with agents. It shouldn't cause overfitting, because each one is looking at a held out sample. So it's not an overfitting issue.

It's quite a subtle issue: Enes through never trained aren't. This sample is from OB bootstrap samples. They also then you're, never gon na grab, 63 % of writes chance to OB is one minus 63 percent exactly yeah both you sure. So then, if you know, why would the score be lower than the validation school and then that you're, leaving sort of like a black hole in the data that there's like there, two points you're never going to sample and I'm not gon na be represented by the Model, ah, no, that's not true, though, because each tree is looking at a different set right so that I won't be so like we've got like, I don't know dozens of models right and a niche one there's a different set of rows, which, which happened to be Held out right, and so when we calculate the oeob score for like let's say row three, we say: okay, row three is in this tree this tree and that's it, and so we calculate the prediction on that tree and for that tree and we'd average. Those two predictions, and so with enough trees, you know each one has a 30 or so percent chance, sorry, forty or so percent chance that the row is in that tree. So if you have fifty trees, it's almost certain that every row is going to be mentioned somewhere. Did you have an idea term, which relevation said we can use the whole forest to make the predictions, but here we cannot use the whole forest, so we cannot exactly see exactly so.

Every road is going to be using a subset of the trees to make its prediction and with less trees. We know we get a less accurate prediction. So that's that's a subtle one right and if you didn't get it have a think during the week. Until you understand why this is because it's a really interesting test of your understanding of random forests, it like why is our B score on average, less good and your validation is for they're both using random subnet subsets anyway, it's really close enough right. So why have a validation set at all when you're using random forests? If it's a randomly chosen, validation set it's not strictly speaking necessary,

but you know you've got like four levels of things to test right, so you could like test on the oeob when that's working. Well, you can test on the validation set, you know, and hopefully, by the time you check against

2. 00:11:35

- **Kaggle Public and Private test sets for Leaderboard,**
- **the risk of using a totally random validation set, rerun the model including Validation set.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

The test set, there's going to be surprises, so that'll be one good reason. Then what cattle do the way they do? This is kind of clever while CAG will do. Is they split the test set into two pieces, a public and a private, and they don't tell you which is rich, so you submit your predictions to cattle and then a random 30 % of those are used to tell you the leaderboard score. But then, at the end of the competition that gets thrown away and they use the other 70 % to calculate your real score. So what that's doing is that you're making sure that you're not like continually using that feedback from the leaderboard to figure out some set of hyper parameters that happens to do well on the public that actually doesn't generalize okay. So it's a great test like this is one of the reasons why it's good practice to use cattle, because at the end of a competition at some point, this will happen to you and you'll drop a hundred places on the leaderboard the last day of the competition. When they use the private test set and say: oh okay, that's what it feels like to overfit and it's much better to practice and get that sense there than it is to do it in a company where there's hundreds of millions of dollars on the line. Okay! So this is like the easiest possible situation where you're able to use a random sample for your validation set.

Why might I not be able to use a random sample from my validation set or possibly fail in the case of something where we're forecasting we can't randomly sample, because we need to maintain the temporal ordering hello? What is that? Because it doesn't, it doesn't make sense. So, in the case of like an ARMA model, I can't use like I can't pull out random rows, because there's I'm thinking that there's like a certain dependency or I'm trying to model a certain dependency that relies on like a specific lag turn. And if I randomly sample those things, then that lag term isn't there for me to okay. So it could be like a technical, modeling issue that, like I'm using a model that relies on like yesterday, the day before and the day before that and if I randomly removed some things I don't have yesterday and my model might just fail. Okay, that's true, but there's a more fundamental issue. You want to pass it to Tyler and it's a really good point, although you know in general we're going to try to build models that are not little more resilient than that, particularly with yet temporal order. We expect things that are close by in time to be related to things close to them so weeks, so we destroy the water like. If, if we destroy the order, we really aren't going to be able to use that this time is close to this other time.

I don't think that's true, because I can pull out a random sample for a validation set and still keep everything nicely ordered well lame reject things in the future, which we would require as much data close to the hand alert. Okay, that's true. I mean we could be like limiting the amount of data that we have by taking some of it out, but my claim is stronger. My claim is that by using a random validation set, we could get totally the wrong idea about our model.

Caribou wan na have a try. So if our data is imbalanced, for example, we can, if you're randomly sampling it, we can only one class in our validation set. So our fitted model - maybe that's true as well, so maybe you're trying to predict in a medical situation. Who's going to die of lung cancer, and that's only one out of a hundred people and we pick out a validation set that we accidentally have nobody that died of lung cancer. That's also true. These are all good niche examples, but none of them quite say like. Why could the validation set just be plain wrong, like give you a totally inaccurate idea of whether this is going to generalize, and so let's talk about, and the closest is, is what Tyler was saying about time closeness in time. The important thing to remember is when you build a model you're, always you always have a systematic error, which is that you're going to use the model at a later time than the time that you built it right, like you're, going to put it into production, by Which time the world is different to the world that you're in now, and even when you're building the model you're using data which is older than today anyway.

Right so there's some lag between the data that you're building it on and the data that it's going to. Actually be used on your life and a lot of the time, if not most, of the time that matters right. So, if we're doing stuff in like predicting who's going to buy toilet paper in New Jersey - and it takes us two weeks to put it in production and we did it using data from the last couple of years and by that time you know things may look Very different right, and particularly our validation, said if we randomly sampled it right and it was like from a four year period, then the vast majority of that data is going to be over a year old right and it may be that the toilet buying habits of Folks in New Jersey may have dramatically shifted. Maybe they've got a terrible recession there now and they can't afford a high-quality toilet paper anymore or maybe they know their paper. Making industry has gone through the roof and suddenly you know they they're buying what's more toilet paper, because it's so cheap or whatever right, so the world changes and therefore, if you use a random sample for your validation set, then you're actually checking. How good are you at predicting things that are totally obsolete now, but how good are you at predicting things that happened four years ago? That's not interesting, okay, so what we want to do in practice, anytime, there's some temper or peace is to instead say assuming that we've ordered it by time all right.

So this is old, and this is new. That's our validation set okay or if we, you know, I suppose actually do it properly. That's how a validation set. That's our test set, make sense right. So here's that training set - and we use that and we try and be able to model that still works on stuff. That's later in time than anything, the model was built on, and so we're not just testing generalization in some kind of abstract sense, but in a very specific time sense, which is it generalizes to the future? Could you pass it to Suraj please? So when we are, as you said, as you said, there is some temporal ordering in the data. So in that case, is it wise to take the entire full data for training, or only a few recent data set for validation, test or training training? Yeah. That's a whole other question all right. So how do you? How do you get the validation set to be good? So I build a random forest on all the training data. It looks good on the training data. It looks good on the oob right and this is actually a really good reason to have OB if it looks good on the OB that it means you're, not overfitting in a statistical sense right like it's, it's working well on a random sample, but then it looks Bad on the validation set, so what happened? Well, what happened was that you, you somehow failed to predict the future. You're only predicted the past and so Suraj had an idea about how we could fix. That would be okay.

Well, maybe we should just train so like maybe we shouldn't use the whole training set. We should try a recent period only and now it on the downside, we're now using less data, so we

can create less rich models on the upside. It's it's more up-to-date data, and this is something you have to play around with most machine learning functions have the ability to provide a weight that is given to each road solve, for example, with a random forest rather than bootstrapping at random. You could have a weight on every row and randomly pick that row with some probability right and we could like say here's our like probability. We could like pick a curve that looks like that, so that the most recent rows have a higher probability of being selected. That can work really well yeah. It's it's something that you have to try and and if you don't have a validation set that represents the future compared to what you're training on you have no way to know which of your techniques are working. How do you make the compromise between amount of data versus recency of data, so what I tend to do is is when I have this kind of temporal issue, which is probably most of the time once I have something that's working well on the validation set. I wouldn't then go and just use that model on the test set, because the thing that I've trained on is now like, but you know, the test set is much more in the future compared to the training set.

So I would then replicate building that model again, but this time I would combine the training and validation sets together. Okay and retrain the model and at that point you've got no way to test against a validation set. So you have to make sure you have a reproducible, script or notebook that does exactly the same steps in exactly the same ways, because if you get something wrong, then you're going to find on the test set that you've you've got a problem. So so what

3. 00:22:15

- **Is my Validation set truly representative of my Test set. Build 5 very different models and score them on Validation and on Test. Examples with Favorita Grocery.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

What I do in practice is, I need to know, is my validation set, a truly representative of the test set. So what I do is I build five models on the training set. I build five models on the training set and I try to have them kind of vary in how good I think they are right and then and then I score them. My five models on the validation set all right and then I also score them on the test set that so I'm not cheating. So I'm not using any feedback from the test set to change my hyper parameters, I'm only using it for this one thing, which is to check my validation set. So I get my five scores from the test set and then I check that they fall in a line. Okay and if they don't, then you're not going to get good enough feedback from the validation set. So keep doing that process until you're getting a line, and that can be quite tricky right. Sometimes the the test set. You know trying to create something: that's as similar to the real-world outcome as possible. It's difficult right and when you're it kind of in the real world, the same is true of creating the test set like the test set, has to be a close to production as possible, so like. What's the actual mix of customers that are going to be using this? How much time is there actually going to be between when you build the model and when you put it in production, how often you're going to be able to refresh the model? These are all the things to think about when you build that test set okay, so you want to say that first make five models on the training data, yeah and then dilute get a straight-line relationship change your validation and death set. You can't really change the test set generally, so this is assuming that the test sets given it changed to change the validation set.

So if you start with a random sample validation set and then it's all over the place - and you realize oh, I should have picked the last two months and then you pick the last two months. It's still going all over the place in your eyes. Oh, I should have picked it, so that's also from the first of the month to the fifteenth of the month and they'll keep going until changing your validation set until you found a validation set, which is indicative of your test, set results, no sort of five models, Like he was started, maybe like just random data and average, and they just make it their own yeah, yeah, yeah yeah, maybe a exactly maybe yeah, I kind of five like not terrible ones, but you want some variety and you also particularly want some variety and like How well they might generalize through time so one that was trained on the whole training set one that was trained on the last two weeks, one that was trained on the last six weeks. One which used, as you know, lots and lots of columns and might have a fit a bit more yeah. So you kind of want to get a sense of like oh, if my validation set fails to generalize temporarily. I'd want to see that if it fell to generalize statistically I'd want to see that sorry, can you explain a bit more detail what you mean by change your validation set, so it indicates the test set like what does that look like so posit? So, let's take the groceries competition where we're trying to predict the next two weeks of grocery sales, so possible validation sets that Terrence and I played with was a random sample.

The last month of data, the last two weeks of data and the other one we tried was same day range one month earlier, so that the test set in this competition was the first to the 15th of August. Sorry, if this 15, that maybe the 15 to the 30th of August, so we tried like a random sample as four years. We tried the 15th of July to the 15th of August. We tried the 1st of August to the 15th of August and we tried the 15th of July to the 30th of July, and so there were four different validation sets we tried and so with random. You know our kind of results were all over the place with last month. You know they were like not bad, but not great. The last two weeks there was a couple that didn't look good, but on the whole they were good and same day range of months. Early they've got a basically perfect line. That's the part, I'm talking right there. What exactly are you comparing it to from the test site? I just confused what you're creating that graph so for each of those. So for each of my so I build five models right. So there might be like just predict the average. Do some kind of simple group mean of the whole data set? Do some group mean, over the last month of the data set, build a read on forests of the whole thing, build a random forest from the last three weeks on each of those I calculate the validation score and then I retrain the model on the whole training Set and calculate the same thing on the test set, and so each of these points now she tells me how about it ago, in the validation set.

How well did it go in the test set, and so if the validation set is useful, we would say every time the validation set improves. The test set should also score should also improve. Yes, so you just said rate ring dreaming rich rings in modeling on training and validations yeah. That was a step I was talking about here. So once I've got the validation score based on just the training set and then retrain it on the train and validation and check against history. Somebody else so just to clarify my test.

4. 00:28:10

- **Why building a representative Test set is crucial in the Real World machine learning (not in Kaggle),**
- **Sklearn make train/test split or cross-validation = bad in real life (for Time Series) !**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Set you mean submitting it to kaibaland and checking the school if it's cattle, then your test set is Carol's leaderboard in the real world. The test set. Is this third data set that you put aside and it's that third data set that having it reflect real world production differences is the most important step in a machine learning project? Why is it the most important step? Because if you screw up everything else that you don't screw up that you're, no, you screwed up right like if you've got a good test set, then you'll know you screw it up, because you screwed up something else and you tested it and it didn't work out And it's like okay you're not going to destroy the company right. If you screwed up creating the test set. That would be awful right because then you don't know. If you've made a mistake right, you try to build a model. You test it on the test set. It looks good, but the test set was not indicative of real-world environment, so you don't actually know if you better destroy the company right now. Hopefully, you've got ways to put things into production gradually, so you won't actually destroy the company, but you'll at least destroy your reputation at work right. It's like Oh Jeremy, tried to put this thing into production and in the first week the cohort we tried it on their sales halved and we're never better give Jeremy machine-learning job again right. But if Jeremy had used a proper test set then like he would have known.

Oh, this is like half as good. As my validation set said. It would be I'll, keep trying right and now I'm not going to get in any trouble. I was actually like. Oh Jeremy is awesome. He identifies ahead of time when there's going to be a generalization problem. Okay, so this is like this is something that kind of everybody talks about a little bit in machine learning classes, but often it kind of stops at the point where you learned that there's a thing in scikit-learn called make test trains flipped and it returns. These things and off you go right, but the fact that like or here's the cross-validation function right, so the fact that these things always give you random samples tells you that, like much, if not most of the time, you shouldn't be using them. The fact that random forest gives you an oo B for free, it's useful, but it only tells you that this generalizes in a statistical sense, not in a practice since right. So then, finally, there's cross-validation right, which outside of class you guys have been talking about a lot which makes me feel

5. 00:31:04

- **What is Cross-Validation and why you shouldn’t use it most of the time (hint: random is bad)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Somebody's been over emphasizing the value of this technique, so I'll explain what cross-validation is and then I explain why you probably shouldn't be using it most of the time so cross validation says: let's not just pull out one validation set, but let's pull out five say. So, let's assume that we're going to randomly shuffle the data first, all right. This is critical right. We first randomly shuffle the data and then we're going to split it into five groups and then for model number. One we'll call this the validation set and we'll call this. The training set: okay and we'll train and we'll check against the validation and we'll get some rmse R squared whatever and then we'll throw that away and we'll call this. The validation set and we'll call this. The training set and we'll get another score we'll do that five times and then we'll take the

average okay. So that's a cross-validation average accuracy. So who can tell me, like a benefit of using cross-validation over a the kind of standard validation set? I talked about before: how could you pass the phone if you have a smokiness, an chosen course validation will make you solve with a data you have yeah, you can use all of the data. You don't have to put anything aside and you kind of get a little benefit as well in that, like you've, now got five models that you could ensemble together each one of used, which used 80 % of the data. So you know sometimes that ensemble lane can be helpful, I'm fun.

Could you tell me like what what could be some reasons that you wouldn't use? Cross-Validation, we have enough data, so we don't not want the validations and to be included in the model. Trainings process like okay yeah, I'm not sure the cross-validation is necessarily polluting the model. What would be a key like downside of cross-validation but like for deepening? If you have learned them P, be chosen, as annual Network will know the pictures it's more likely to predicative, as is right so sure, but if we, if we put aside some data each time in the cross-validation, can you pass it to Suraj, I'm not so worried About, like I don't think, there's like one of these validation sets is more statistically accurate, yes, Suraj Steven, will you be all fitting together late? I think that's what fun was worried about. I don't see why that would happen like each time we're fitting a model just 100. Each time we're fitting a model. We are absolutely holding in 20 percent of the sample right. So, yes, the five models between them have seen all of the data, but but it's kind of like a random forest independence, is a lot like a random first, each model has only been trained on a subset of the data. Yes, you should see. Please David largely received like it is deep load of time. Oh yes, exactly right, so we have to fit five models rather than one.

So here's a key downside number one it's time, and so, if we're doing deep learning - and it takes a day to run suddenly it takes five days or we need five GPUs. Okay. What about my earlier issues about validation sets Jona pass it over there. What's remaining was a so if you had like temporal data, wouldn't you be like shuffling when you e breaking that relation? Well, we could unravel it afterwards. We could reorder it like. We could shuffle get the training set out and then sort it by time like and like this, presumably there's a date column there. So I don't think I don't think it's going to stop us from building a model. Did you have with cross-validation your building? Five, even validation sets, and if there is some sort of structure that you're trying to capture in your validation sets of Mary, your test set you're, essentially just throwing that a chance to construct that yourself right. I think you're gon na say that I think you said the same thing as I'm gon na say, which is, which is that our earlier concerns about why random validation sets are a problem are entirely relevant here. Well, these validation sets a random. So if a random validation set is not appropriate for your problem most likely because, for example, of temporal issues, then none of these four validation set of five validation sets are any good they're all random right.

And so, if you have temporal data like we did here, there's no way to do cross, validation really or like probably no good way to do cross validation. I mean you're wan na, have your validation set be as close to the test set as possible, and so you can't do that by randomly sampling, different things. So so, as fone said, you may well not need to do cross validation because, most of the time in the real world, we don't really have that little data right unless your data is based on some very, very expensive labeling process or some experiments that take a Look cost a lot to run or whatever, but nowadays that's data. Scientists are not very often doing that kind of work. Some um, in which case this is an issue it must have assigned. So we probably don't need to, as nishan said, if we do do it, it's going to take a whole lot of time all right and then, as earnest said, even if we did do it and we took up all that time. It's

like it was totally the wrong answer, because random validation sets are inappropriate for a problem. Okay, so I'm not going to be spending much time on cross validation, because I just I think it's an interesting tool to have it's easy to use. Sosuke learn has a cross validation thing. You can go ahead and use, but it's it's. It's not that often that it's going to be an important part of your toolbox. In my opinion, you'll come up some points. Okay, so that is validation tips.

So then the other thing we started talking about

6. 00:38:04

- **Tree interpretation revisited, lesson2-rf_interpreter.ipynb, waterfall plot for increase and decrease in tree splits,**
- **'ti.predict(m, row)'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Last week and got a little bit stuck on because I screwed it up was tree interpretation, so I'm actually going to cover that again without the error and dig into it in a bit more detail. So can anybody tell me what tree interpreter does and how it does it? What do you remember it's a difficult one to explain. I don't think I did a good job of explaining it. So don't worry if you don't do a great job, but does anybody wan na have a go explaining it? Well? Okay, that's fine! So let's start with the output of tree interpreter. So if we look at a single model, a single tree, in other words here, is a single tree. Okay, so to remind us, the top of a tree is before there's been any split at all. So ten point one: eight nine is the average log price of all of the options in our training set. So I'm going to go ahead and draw right here, ten point: one: a nine eight nine is the average of all okay and then, if I go a couple of system less than or equal to 0.5, then I get ten point three, four, five. Okay. So, for this subset of 16,800 coupler is less than or equal to point five. The average is ten point three four five and then off the people with a couple of system less than or equal to point 5. We then take the subset, we're enclosure at less than or equal to two and the average there of rob.

Sale price is nine point: nine, five, five against nine point: nine five and then final step in our tree as model ID just for this group with no capitalist system with enclosed lesson or then let's just take model ID less than or equal to forty five. Seventy three - and that gives us ten point, two, two six. Okay, so then we can say all right, starting with ten point one: oh nine one, eight nine average for everybody in our training set for this particular tree, subsample of twenty thousand, adding in the capital decision or couple or less than a two point. Five increased. Our prediction by point one: five: six, so if we predict it with a naive model of just the mean that would have been ten point when I known adding in just the coupler decision, would have changed it to ten point three, four five. So this variable is responsible for a point: one: five: six increase in a prediction from that: the enclosure no decision was responsible for a point. Three nine five decrease the model. Id was responsible for eight point: two, seven, six increase until eventually that was our final decision. That is our prediction for this option of this particular sale price. So we can draw that as what's called a waterfall block right and what our four plots are. One of the most useful plots.

I know about and weirdly enough, there's nothing in Python to do them, and this is one of these things where there's this disconnect between, like the world of management, consulting and business, where everybody uses water for plots all the time and like academia who have

no idea. What these things are, but like every time like you're looking at say here is last year's sales for Apple, and then there was a change in that iPhones increased by this amount max decreased by that amount and iPads increased by that amount. Every time you have a starting point in a number of changes and a finishing point, waterfall charts are pretty much always the best way to show it. So here our prediction for price based on everything, ten point: one: eight nine there was an increased blue means increase of 0.156, the coupler decrease of 0.395 or enclosure increase model ID of point two, seven six so decrease, but I increase decrease increase to get to our Final ten point: two: six: six, so you see how what a porch light works so with Excel 2016 its built-in you just click, insert waterfall chart and there it is. If you want to be a hero, create a waterfall chart package format, plot lab, put it on pip and everybody will love you for it. There are some like really crappy, gist's and manual notebooks and stuff around.

These are actually super easy to build, like you, basically do a stacked column plot, where the the bottom of this is like all white right like you can kind of do it, but if you can wrap that up or all and put the data the points in The right spots and color them nicely that would be totally awesome. I think you've all got the skills to do it and would make you know, be a terrific thing for your portfolio. So there's an idea could make an interesting cattle Colonel even like here's. How to build a waterfall plot can scratch and by the way I've been putting this up on yep, you can all use it. So in general, therefore, obviously going from the all and then going through each change, then the some both all of those is going to be equal to the final prediction. So that's how we could say if we were just doing a decision tree, then you know you're. Coming along and saying like how come this particular option, was this particular price? And it's like: well your prediction for it and like oh, it's because of these three things had these three impacts right so for a random forest. We could do that across all of the trees right. So every time we see coupler, we add up that change. Every time we see enclosure, we add up that change. Every time we see model, we add up that change, okay, and so then we combine them all together.

We get what tree interpreter does right, so you could go into the source code for tree interpreter right and it's not at all complex logic or you could build it yourself and you can see how it does exactly this. So when you go tree and predict with a random forest model for some specific option, so I've got a specific row here. This my zero index row. It tells you okay, this is the prediction the same as the random forest prediction bias. This is going to be always the same. It's the average sale price for for everybody for each of the random samples in the tree, and then contributions is the average of all so the total of all our contributions for each time. We see that specific column appear in a tree right. So last time I made the mistake of not sorting this correctly. So this time MP, dot arc sort, is a super handy function at sorts. It doesn't actually sort contribution zero. It just tells you where each item would move to if it were sorted so now by passing ID access to each one of the column, the level contribution. I can then print out all those in the right order. So I can see here, here's my column. Here's the level and the contribution, so the fact that it's a small version of this piece of industrial equipment meant that it was less expensive right, but the fact that was made pretty recently meant that was more expensive.

The fact that it's pretty old, however, made that it was less expensive right, so this is not going to really help you much at all with like a cattle styled situation where you just need predictions, it's going to help you a lot in a production environment or Even pre production right so like something which any good manager should you should do. If you say here's a machine learning model, I think we should use as they should go away and grab a few

examples of actual customers or actual options or whatever and check whether your model looks intuitive, alright and if it says, like my prediction, is that you Know lots of if people are going to really enjoy this crappy movie. You know it is like wow that was a really crappy movie. Then they're going to come back to you and say like explain why your models telling me that I'm going to like this movie, because I hate that movie and then you can go back and you say well, it's because you like this movie and because you're this Age range and you're this gender on average. Actually people like you did like that movie: okay, yeah. What's the second element of each temple, this is saying for this particular row. It was a mini and it was 11 years old and it was a hydraulic excavator track. 3 to 4 metric tons, so it's just feeding back and telling you it's, because this is actually what it was. It was these numbers, so I just went back to the original data to actually pull out the descriptive versions of each one.

Okay. So if we sum up all the contributions together and then add them to the bias, then that would be the same as adding up those three things: adding it to this and, as we know from our waterfall chart, that gives us our final prediction. This is a almost totally unknown technique and this particular library is almost totally unknown as well. So like it's. A great opportunity to you know show something that a lot of people like it's totally critical in my opinion, but but rarely none. So that's

7. 00:48:50

- **Dealing with Extrapolation in Random Forests,**
- **RF can’t extrapolate like Linear Model, avoid Time variables as predictors if possible ?**
- **Trick: find the differences between Train and Valid sets, ie. any temporal predictor ? Build a RF to identify components present in Valid only and not in Train ‘x,y = proc_df(df_ext, ‘is_valid’)’,**
- **Use it in Kaggle by putting Train and Test sets together and add a column ‘is_test’, to check if Test is a random sample or not.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

That's kind of the end of the random forest interpretation piece and hopefully, you've now seen enough that when somebody says we can't use modern machine learning techniques because they're black boxes that are interpretive all you have enough information to say: you're full of right, like they're, extremely Interpretable and the stuff that we've just done, you know trying to do that with a linear model. Good luck to you, you know, even where you can do something similar with a linear model trying to do it. So that's not giving you totally the wrong answer and you had no idea it's a wrong answer. It's going to be a real challenge, so the last step we're going to do before we try and build up our own random forest is deal with this tricky issue of extrapolation. So in this case, if we look at our tree, let's look at the accuracy of our most recent trees. We still have you know a big difference between our validation score and our training score the actually, in this case, it's not too bad - that the difference between the oob and the validation is actually pretty close. So if there was a big difference between validation and olb, like I'd, be very worried about that, we've dealt with the temporal side of things correctly. Let's just have a look at, I think the most recent model here it was yeah, so there's a tiny difference right and so on Tagle at least you kind of need that last decimal place in the real world.

I probably stopped here, but quite often you'll see. There's a big difference between your validation score and your OB score. Now I will show you how you would deal with that, particularly because actually we know that the the oeob should be a little worse because it's using this less trees. So it gives me a sense that we should get to do a little bit better and so the reason we the way we should be able to a little bit better, is by handling the time component a little bit better. So here's the problem with random forests when it comes to extrapolation when you, when you've got a data set. That's, like you know, four got four years of sales data in it and you create your tree right and it says like oh if these, if it's in some particular store and at some particular item - and it is on special, you know: here's the average price and It actually tells us the average price you know over the whole training set, which could be pretty old right, and so when you then want to step forwards like what's going to be the price next month, it's never seen next month and and where else, with a Kind of a linear model: it can find a relationship between time and price where, even though we only had this much day when you then go and predict something in the future, it can extrapolate that, but a random forest can't do that.

There's no wave, if you think about it, for a tree to be able to say well next month, it would be higher still so there's a few ways to deal with this and we'll talk about it over the next couple of lessons. But one simple way is just to try to avoid using time variables as predictors. If there's something else, we could use that's going to give us a better. You know something that kind of a stronger relationship, that's actually going to work in the future. So in this case, what I wanted to do was to first of all figure out: what's the difference between our validation set and our training set like if I understand that difference between our validation set and our training set, then that tells me what are the predictors, Which which have a strong temporal component and therefore they may be irrelevant by the time I get to the future time period. So I do something really interesting, which is I create a random forest where my dependent variable is? Is it in the validation set right? So I've gone back and I've got my whole data frame with the training and validation altogether and I've created a new column Court is valid which I've set to one and then for all of the stuff in the training set. I set it to zero. That's about a new column which is just is this in the validation set or not, and then I'm going to use that as my dependent variable and build a random first. So this is a random forest not to predict price.

That protect is this in the validation set or not, and so, if your variables were not time dependent, then it shouldn't be possible to figure out if something's, in the validation set or not. This is a great trick in cattle records in cattle. They often won't tell you whether the test set is a random sample or not. So you could put the test set and the training set together, create a new column called, is test and see. If you can predict it, if you can, you don't have a random sample, which means you have to come and figure out how to create a validation set from it right, and so, in this case I can see I don't have a random sample, because my validation Set can be predicted with a 0.9999 r-squared, and so then, if I look at future importance, the top thing is sales ID, and so this is really interesting. It tells us very clearly. Sales ID is not a random identifier, but probably it's something. That's just set consecutively as time goes on. We just increase the sales ID so elapsed. That was the number of days since the first date in our data set. So not surprisingly, that also is a good predictor. Interestingly machine ID. Clearly, each machine is being labeled with some consecutive identifier as well, and then there's a big. Don't just look at the order. Look at the value so 0.7. 0.1. 0.0. 7.00. Okay, stop right! These top three are hundreds of times more important than the rest right.

So let's next grab those top three right and we can then have a look at their values both from

the training set and in the validation set, and so we can see. For example, sales ID on average is divided by thousand on averages: 1.8 million in the training set and 5.8 million in the validation set right. So you'd like you, can see just confirm like okay they're very different, so let's drop them okay. So after I drop them. Let's now see, if I can predict whether something's in the validation set, I still can with 0.98 pass quit. So once you remove some things, then other things can like come to the front and it now turns out. Okay, that's not surprisingly age. You know things that are old. Are you know more likely, I guess to be in the validation set, because there's you know earlier on in the training set, yet they can't be old. Yeah yeah made same reason. So then we can try removing those as well and so once we let's see where do we go up here, yeah, so what we can try doing is. We can then say: alright, let's take the saleslady. So, let's machine ID from the first one, the age year, made sale day of year from the second one and say: okay: these are all time dependent features, so I still want them in my random forest if they're important right, but if they're not important, then taking Them out, if there are some other long-term dependent variables that that work just as well, that would be better right, because now I'm going to have a model that generalizes over time better. So here I'm just going to go ahead and go through each one of those features and drop each one, one at a time: okay, retrain, a new random forest and print out the score okay.

So before we do any of that, our score was point. Eight eight for our validation versus point eight 900 B, and you can see here when I remove sales ID. My score goes up and this this is like what we're hoping for we've removed a time dependent variable. There were other variables that could find similar relationships without the time dependency, so removing it cost our validation to go up now. Oob didn't go up right because this is genuinely statistically you're useful, predictor right, but it's a time dependent one when we have a time dependent, validation set. So this is like really subtle, but it can be really important right. It's trying to find the things that gives you a generalizable time across time prediction and here's how you can see it. So it's like okay, we should remove sales ID for sure right, but sale elapsed didn't get better okay, so we don't want that machine. Id did get better right from eight eight, eight to eight nine three right. So it's actually quite a bit better age got a bit better. You made got worse. Sale day of year, got a bit better okay. So now we can say alright, let's get rid of the three where we know that getting rid of it actually made it better. Okay and, as a result, look at this we're now up to nine one five. Okay, so we've got rid of three time dependent things and now, as expected, validation is better than our Obi okay.

So that was a super successful

8. 00:59:15

- **Our final model of Random Forests, almost as good as Kaggle #1 (Leustagos & Giba)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Approach there right and so now we can check the feature importance and let's go ahead and say all right. That was pretty damn good. Let's now leave it for a while, so give it a hundred and sixty trees, don't show it and see how that goes. Okay, and so, as you can see like, we did all of our interpretation, all of our fine-tuning, basically with smaller models subsets and at the end we run the whole thing. You actually still only took 16 seconds, and so we've now got an RMS see of 0.21. Okay, so now we can check that against cattle again we can't we.

Unfortunately, this older competition we're not allowed to enter anymore to see how he would have gone. So the best we can do is check whether it looks like we could have done we're all based on their validation set, so it should be in the right area and yeah. Based on that, we would have come first. Okay, so you know. I think this is an interesting series of steps right, so you can go through the same series of steps in your cattle projects and, more importantly, your real-world projects. So one of the challenges is once you leave this learning environment, suddenly you're, surrounded by people who they they've had not have enough time. They always want you to be in a hurry. They're, always telling you you know do this and then do that. You need to find the time to step away right and go back, because this is a genuine real-world, modeling process you can use and it gives, when I said, kids world-class results.

I mean it right like this guy who won this lista costs. Sadly, he's passed away, but he is the top kaggle competitor of all time like he. He won, I believe, like dozens of competitions, so we can get a score even within cuy of him, then we are doing really really well okay. So, let's take a five-minute break and we're going to come back and build our own random. First, I just wanted to clarify something quickly. A very good point during the break was going back to the change in R squared between here, and here it's not just due to the fact that we removed these three predictors. We also went reset our F samples right, so they actually see the impact of just removing. We need to compare it to the final step earlier, so it's actually compared to 907. So removing those three things took us from 907 nine one, five. Okay, so I mean - and you know in the end of course, what matters is our final model that yep just to clarify? Okay, so um? Some of you have asked me about writing your own random forests from scratch. I don't know if any of you have given it a try. Yet my original plan here was to do it in real time and then, as I started to do it, I realized that that would have kind of been boring because to you, because I screw things up all the time. So, instead we might do more of like a walk through the code together. Just as an aside, this reminds me talking about the exam hammock.

She somebody

9. 01:03:04

- **What to expect for the in-class exam**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Asked on the forum about like what what can you expect on the exam? The basic plan is to make it a exam, be very similar to these notebooks, so it'll probably be a notebook that you have to you know, get a data set, create a model. Trainer feature importance whatever right and the plan is that it'll be open, block open Internet. You can use whatever resources you like. So basically, if you're entering competitions, the exam should be very straightforward. I also expect that there will be some pieces about like here's. A partially completed random forest or something you know finish finish, writing this step here or here's a random forest implement feature importance or in you know, implement one of the things we've talked about, so it open it. You know the exam will be much like what we do in class and what you're expected to be doing during the week. There won't be any define this or tell me the difference between this word and that word or whatever, there's not going to be any rote learning it'll be entirely like. Are you an effective machine learning practitioner ie can use the algorithms do you know? Can you create an effective validation set and can you can you

create parts of the algorithm implement them from scratch? So it'll be all about writing code. Basically, so if you're not comfortable writing code to practice machine learning, then you should be practicing that all the time, if you are comfortable, you should be practicing that all the time, also whatever you're doing write code to implement random to do machine learning.

Okay, so I I kind of have a particular way of writing code and I'm not going to claim it's the only way of writing code, but it might be a little bit different to what you're used to and hopefully you'll find it at least interesting. Creating implementing random forest

10. 01:05:04

- **Lesson3-rf_foundations.ipynb, writing our own Random Forests code.**
- **Basic data structures code, class 'TreeEnsemble()', np.random.seed(42)' as pseudo random number generator**
- **How to make a prediction in Random Forests (theory) ?**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Algorithms is actually quite tricky, not because the codes tricky like generally speaking, most random first algorithms are pretty conceptually easy at all that, generally speaking, academic papers and books have a knack of making them look difficult, but they're not difficult. Conceptually, what's difficult is getting all the details right and knowing and knowing when you're right, and so in other words, we need a good way of doing testing. So if we're going to reimplemented a we want to create a random forest in some different framework, different language, different operating system, you know I would always start with something that does exist right. So in this case, we're just going to do is learning its exercise. Writing a random forest in Python, so for testing, I'm going to compare it to an existing random forest implementation. Okay, so that's like critical anytime you're doing anything involving like non-trivial amounts of code and machine learning, knowing whether you've got it right or wrong is kind of the hardest fit. I always assume that I've screwed everything up at every step, and so I'm thinking like okay, assuming that I screwed it up. How do I figure out that I screwed it up right and then much to my surprise from time to time I actually get something right and then I can move on okay, but most of the time I get it wrong.

So, unfortunately, with machine learning there's a lot of ways you can get things wrong that don't give you an error, they just make your result like slightly less good, and so that's that's what you want to pick up so, given that I want to kind of compare It to an existing implementation, I'm going to use our existing data, set our existing validation set and then to simplify things. I'm just going to use two columns to start with, so let's go ahead and start writing a random forest. So my way of writing nearly all code is top-down, just let my teaching, and so, if I top-down I start by assuming that everything I want already exists, so in other words the first thing I want to do, I'm going to call this a tree ensemble right. So, to create a random forest. The first question I have is: what do I need to pass in right? Why I need to initialize my random first, so I'm going to need some independent variables, some dependent variable pick how many trees I want I'm going to use the sample size parameter from the start here. So how big you want each sample to be, and then maybe some optional parameter of. What's the smallest leaf size, okay, for testing, it's nice to use a constant random seed, so we'll get the same result each time. So this is just how you set a random seed. Okay, maybe it's worth mentioning is for those of you, unfamiliar with it.

Random number generators on computers aren't random at all, now, actually constitute a random number generators and what they do is given some initial starting point. In this case 42, a pseudo-random number generator is a mathematical function that generates a deterministic, always the same sequence of numbers such that those numbers are designed to be as uncorrelated with the previous number as possible: okay and as unpredictable as possible and as uncorrelated as possible. With something with a different random seed, so the second number in in the sequence, starting with 42, should be very different to the second number, starting with 41 and generally they involve kind of like taking. You know you know using big prime numbers and taking mods and stuff like that, it's kind of an interesting area of math. If you want real random numbers, the only way to do that is again. You can actually buy hardware called a hardware. Random number generator that'll have inside them like a little bit of some radioactive substance and and like something that detects how many things it's spitting out or you know there will be some hardware thing any current system time is. Is it a valid random, like random number generation, no sense, so that would be for maybe for a random seed right. So this thing of like what do we start the function with so one of the really interesting areas is like in your computer.

If you don't set the random seed, what is it set to and yeah? Quite often, people use the current time for security, like obviously, we use a lot of random number stuff for security stuff like if you're generating an SSH key, you need some. It needs to be random, it turns out, like you know, people can figure out roughly when you created a key like they could look at like oid RSA has a timestamp and they could try. You know all the different nanoseconds starting points for a random number generator around that time, step and figure out your key. So in practice, a lot of like really random high randomness requiring applications actually have a step. That say, please move your mouse and type random stuff at the keyboard for a while, and so it like gets you to be a sort. That's called entropy to be a source of entropy. Other approaches is they'll. Look at, like you know the hash of some of your log files, or you know stuff like that. It's a really really fun area. So, in our case our purpose actually is to remove randomness. So we're saying: okay generate a series of pseudo-random numbers starting with 42. So it always should be the same. So if you haven't done much stuff in Python oo, this is a basically standard idiom. At least I mean I write it this way most people don't, but if you pass in like 1, 2 3 4 5 things that you're going to want to keep inside this object, then you basically have to say self dot x equals x, self dot y equals Y self, that sample equals sample right, and so we can assign to a tuple from at a port.

So you know okay. This is like my way of coding. Most people think this is horrible, but I prefer to be able to see everything at once, and so I know in my code anytime, I see something that looks like this. It's always all of the stuff in the method being set. If I did it a different way than half the codes now come off the bottom of the page, and you can't see it all right so um, so that was the first thing I thought about is like okay to create a random forest. What information do you need, then? I'm going to need to store that information inside my object, and so then I need to create some treats. I had a random forest is something that creates and is something that has some trees. So I fit basically figured okay list. Comprehension to create a list of trees, how many trees do we have or you put n trees trees? That's what we asked for so range entries gives me the numbers from 0 up to n trees. Minus 1. Ok! So if I create a list comprehension that lips through that range calling create tree each time, I now have entries trees and also, I add, to write that I didn't have to think at all, like that's all like obvious and so I've kind of delayed. The thinking to the point where it's like well wait: we don't have something to create a tree. Okay, no worries, but let's pretend we did if we did. We've now created a random forest. Okay we'd still need to like do a few things.

On top of that, for example, once we have it, we need a predict function.

So, okay! Well, let's write a prediction function. How do you predict in a random forest? Can somebody tell me either based on their own understanding or based on this line of code? What would be like your one or two-sentence answer? How do you make a prediction in a random forest, positive, Spencer uh? You would want to over every tree for your like the row that you're trying to predict on average the values that your that each tree would produce for that book. Good, and so you know that's a summary of what this says right so for a particular row. Right or maybe this is a number of rows - go through each tree calculate its prediction. So here is a list comprehension that is calculating the prediction for every tree for X. I don't know if X is one row or multiple rows, it doesn't matter right as long as as long as trade, I predict works on it and then once you've got a list of things, a cool trick to know is you can pass numpy dot mean a Regular non mum pie list, okay and it'll - take the mean you just need to tell it: access equals zero means everage it across the lists. Okay, so this is going to return the average of that predict for each tree, and so I find list comprehensions. Allow me to write the code in the way that the brain was like. You could take the word Spencer said and like translate them into this code, or you could take this code and translate them into words.

Like the one Spencer said right, and so when I write code, I want it to be as much like that as possible. All right, I want it to be readable and so hopefully you'll find like. When you look at the past AI code, you can understand how to journey through X. I try to write things in a way that you can read it and like it kind of turn it into English in your head. So if I say correctly, that predict method is recursive, it's no, it's calling trade predict and we haven't written a tree. Yet so self trees is going to contain a tree object. So this is tree. Ensemble dot predict and inside the trees is a tree, not a tree ensemble. So this is called an trade product, not tree ensemble, dotted it the question. Okay, so we nearly finished riding around and for our seventh week. All we need to do now is write, create tree right, so um, based on this code here or on your own understanding, of how we create trees in a random forest. Can somebody tell me, let's take a few seconds? Have a raid have to think and then I'm going to try and come up with a way of saying how do you create a tree in a random forest? Okay, who wants to tell me yes, okay, let's tireless work cluster, you take your you're, essentially taking a random sample or of the original data and then you're just it just constructing a tree. However, that happens so construct a decision tree like a non random tree from a random sample of the data. Ok, so again, like we've delayed any actual thought process.

Here, we've basically said: ok, we could pick some random IDs. This is a good trick to know. If you call NP random permutation passing in an inch it'll give you back a randomly shuffled sequence from zero to that inch right, and so then, if you grab the first n items of that, that's now a random subsample. So this is not doing bootstrapping we're not doing sampling with replacement here, which i think is fine. You know for my random forest, I'm deciding that it's going to be something where we do subsampling, not bootstrapping, ok, so here's a good line of code to know how to write because it comes up all the time like I find in machine learning most algorithms. I use are somewhat random and so often I need some kind of random sample. Can you pass that tartaric entry? Won't they give you 1 1 extra, because the easier it will go from 0 to length? No. So this will give you if lens self dot y is of size n. This will give you n a sequence of length, n, so 0 to n minus 1 and then from that I'm picking out self dot sample size, so the first sample size. Ladies, I have a comment on bootstrapping. I think this method is better because we have transfer giving more weights to each observation, or am I thinking wrong? I think you proposed wrapping. We could also give weights, I mean we single observations more than they are like without one thing that weights, because I'm

bootstrapping with with replacement, we can have a single observation and dr. pitz of it yeah the same tree yeah.

It just feel weird, but I think the actual theory or empirical results backs up higher intuition that it's worse it'd be interesting. To look look back at that. Actually, personally, I prefer this because I feel like most of the time we have more data than we want to put a tree at once. I feel like back when Bremen created random forests. It was 1999, it was kind of a very different world. You know where we pretty much always wanted to use all the data we had, but nowadays I would say: that's generally not what we want. We normally have too much data, and so what people tend to do is they'll like fire up a spark cluster and they'll run it on hundreds of machines when it makes no sense, because if they had just used a subsample each time they could have done it. On one machine and like the the overhead of like spark, is a huge amount of i/o overhead, like I know you guys are doing distributed computing now, if you, if you've, looked at some of the benchmarks, yeah yeah exactly so, if you do something on a single Machine, it can often be hundreds of times faster, because you don't have all this this i/o overhead. It also tends to be easier to write. The algorithms like you can use like SK, learn easier to visualize and cheaper, so forth. So, like I almost always avoid distributed computing and I have my whole life like even 25 years ago, when I was studying in machine learning, I you know still didn't use clusters because I so I always feel like whatever I could do with a cluster now I Could do with a single machine in five years time, so one of us focus on always being as good as possible with the single machine. You know and that's going to be more interactive and more iterative and work for me.

So, ah, okay, so so again, we've like delayed thinking to the point where we have to write decision tree, and so hopefully you get an idea that this top-down approach, that goal is going to be that we're going to keep delaying thinking so long that that we Delay it forever like, like eventually we've somehow written the whole thing without actually having to think right, and that's that's kind of what I need is I'm kind of slow right. So this is why I write code this way and notice, like you, never have to design anything in. You just say hey what, if somebody already gave me the exact API I needed, how would I use it? Okay and then and then okay to implement that next stage,

11. 01:21:04

- **class 'DecisionTree()',**
- **Bonus: Object-Oriented-Programming (OOP) overview, critical for PyTorch**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

What would be the exact API? I would need to implement that right. You keep going down until eventually you're like oh, that already exists. Okay, so this assumes we've got a class port decision tree, so we're going to have to create that so a decision tree is something so we already know what we're going to have to pass it because we just passed it right. So we're passing in a random sample of X's a random sample of Y's um uhh indexers is actually so. We know that down the track, so I've got a plan a tiny bit. We know that a decision tree is going to contain decision trees which themselves contain decision trees, and so, as we go down the decision tree there's going to be some subset of the original data that we've kind of got, and so I'm going to pass in the Indexes of the data that we're actually going to use here - okay, so initially it's the entire random sample all right, so I've got the whole team. I've got the

whole range and I turn that into an array. So that's 0, the indexes from 0 to the size of the sample and then what is passed down the mean left side. So everything that we got for constructing the random forest where to pass down the decision tree except, of course, num trees, which is irrelevant for the decision tree so again, now that we know that's the information we need, we can go ahead and store it inside this Object, so I'm pretty likely to need to know how many rows we have in this tree, which I generally call n.

How many columns do I have, which I generally call C. So the number of rows is just equal to the number of indexes well given, and the number of columns is just like. However, many columns there are in our independent variables, so then we're going to need this value here. We need to know for this tree. What's its prediction right, so the prediction for this tree is the mean of our dependent variable or those indexes which are inside this part of the tree, alright, so at the very top of the tree, it contains all the indexes right, I'm assuming that by the time We've got to this point. Remember: we've already done the random sampling right. So when we talk about indexes we're not talking about the random sampling to create the tree, we're assuming this tree now has some random sample inside decision tree. This is this: is the one of the nice things right inside decision tree hole, random sampling, things gone right that was done by the random forest right. So at this point, we're building something. That's just a plain old decision tree. It's not in any way a random sampling, anything, it's just a plain old position tree right, so the indexes is literally like which subset of the data that we got to so far in this tree and so at the top of the decision tree. It's all the data right, so it's all of the indexes, okay, so all of the indexes.

So this is therefore all of the dependent variable that are in this part of the tree, and so this is the value mean of that. That makes sense anybody could be any questions about about that. So, ah yes, hey, pastor, Chen Qi actually, just to let you know, there's a large portion of us don't have a over B. I mean all P experiments, okay, yeah sure, so so, quick so quick over P prenup would be helpful, great yeah, okay, who is done object-oriented programming in some programming, language, okay, so you've all used actually lots of object-oriented programming in terms of using existing classes right. So every time we've created a random forest, we've called the random forests constructor and it's returned an object and then we've called methods and attributes on that object. So fit is a method you can tell, because it's got parentheses after it right where else yeah, oh I'll, be score, is a property or an attribute doesn't have parentheses after it? Okay, so inside an object there are kind of two kinds of things there: the functions that you can call. So you have object, dot, function, parentheses, arguments or there are the properties or attributes you can grab, which is object, dot and then just the attribute name, no parentheses. So when and then the other thing that we do with objects, is we create them? Okay, we pass in the name of a class and it returns us the object and you have to tell it all of the parameters necessary to get constructed.

So let's just copy this code and see how we're going to go ahead and build this. So the first step is we're not going to go. N equals random forest regressor. We're going to go M equals tree ensemble we're creating a classical tree ensemble and we're going to pass in various bits of information. Okay, so maybe we'll have ten trees sample size of a thousand or maybe a min leaf of three okay, and you can always like choose to name your admits or not. So when you've got quite a few, it's kind of nice to name them so that just so we can see what each one means. It's always optional. So we're going to try and create a class that we can use like this and then the notional we're going to bother with dot fit because we've passed in the X and the y right like in scikit-learn. They use an approach where, first of all, you construct something without telling it what they did here is, and then you pass in the day we're doing these two

steps at once. We're actually passing in the data right and so then, after that we're going to be going and dot so we're going to go. Creds equals m predict passing in maybe some validations there. Okay, so we that's that's the API, we're kind of creating here. So this thing here is called a constructor, something that creates an object is called a constructor and Python. There's a lot of ugly hideous things about Python, one of which is they it uses.

These special magic method, names underscore underscore init underscore underscore - is a special magic method. That's caught, it's called when you try to construct a class. So when I call tree ensemble parentheses, it actually calls tree ensemble dot, they see people say dunder init, I kind of hate it, but anyway timed it. You know double underscore in it double underscore dunder init. So that's why we've got this method called dunder, init. Okay, so when I call tree ensemble is going to call this method, another hideously ugly thing about pythons oo. Is that there's this special thing where, if you have a class and to create a class, you just wrecked class in the name of us all of its methods, automatically get sent one extra parameter when extra arguments, which is the first argument - and you can call it Anything you like, if you call it anything other than self everybody will hate you and you're a bad person, so call it anything you like as long as it's self. So so that's why you always see this, and in fact I can immediately see here. I have a bug, anybody see the bug in my predict function. I should have so right. I like it always do it right. So anytime. You try and call a method on your own class and you get something saying you're passed in two parameters and it was only expecting one you forgot so okay so like this is a really dumb way to add.

Oh okay to a programming language, but the older languages like Python often did this because they kind of needed to they started out not being oo and then they kind of added oo in a way that was hideously ugly, so Perl, which predates plaything by a little Bit kind of, I think, really came up with this approach and unfortunately other languages of that era stuck with it. So you have to add in this magic self, so the magic self. Now, when you're inside this class, you can now pretend, as if any property name you like exists, so I can now pretend there's something called self dot X. I can read from it. I can write to it right, but if I read from it - and I haven't yet written to it, I'll get an error, so the stuff that's passed to the constructor gets thrown away by default. Like there's nothing that like says you need to rip, this class needs to remember what these things are, but anything that we stick inside self is remembered for all time. You know, as long as this object exists, you can access it. It's remembered so now that I've gone. In fact: let's do this right so that let's create the tree ensemble class and let's now instantiate it. Okay, of course, we haven't got X. We need to call X, train y trade. Ok decision tree is not defined. So, let's we had a really minimal decision tree there we go okay, so here is enough to actually instantiate our tree ensemble. Okay, so we have to find the inert for it. We have to find the inert for decision tree.

We need decision trees in it to be defined because inside our ensemble in it they're called self directory and then self create tree called the decision tree constructor and then decision tree constructor basically does nothing at all other than save some information right. So at this point we can now go m dot. Okay, and if I press tab at this point, can anybody tell me what I would expect to see press it to Taylor tension? Could you possibly say like we would see a drop-down of all available methods for that class? Okay, it would be in this case. So if M is a tree ensemble, we would have create tree and predict okay, anything else. What oh yeah, as well as earnest, whispered to variables as well yeah, so that the variable could made a lot of things well attributes, so the things that we put inside self. So if I hit tab right there, they are right, as Taylor said, there's create tree. There's predict and then there's everything else we put inside so all right. So if I look at m dot min leaf, if I hit shift enter, what will I see yep? The

number that I just put there I put in leaf is three so that went up the air dam in leaf. This here is a default argument such as, if I don't pass anything it'll be five, but I did pass something right, so three self dot min leaf here. Is it going to be equal to min leaf yeah, so something which like because of this rather annoying way of doing oh, oh, it does mean that it's very easy to accidentally forget so do that right? So if I don't assign it to self dot min leaf right, then I get an error, and so here tree ensemble doesn't happen in leaf right.

So how do I create that attribute? I just put something in it: okay, so if you want to like, if you don't know what a value of it should be yet, but you kind of need to be able to refer to it, you can always feel like self dot min leaf equals. None! That's at least there's something you can read check for numbness and not have an error great now, interestingly, I was able to instantiate tree ensemble, even if I predict refers to a method of decision tree that doesn't exist, and this is actually something very nice about the Dynamic nature of Python is that because it's not like compiling it, it's not checking anything unless you're using it right. So we can go ahead and create decision to predict later and then our our instantiated object will magically start working right. It doesn't actually look up that functions that methods details until you use it, and so it really helps with top-down programming. Okay, so when you're inside a class definition, in other words you're at that indentation level, you know indented one in so these are all class definitions. Any function that you create, unless you do some special things that we're not going to talk about yet is automatically a method of that class, and so every method of that class magically gets a self passed to it. So we could call since we've got a tree.

Ensemble, we could call em create tree and we don't put anything inside those parentheses, because the magic self will be passed and the magic self will be whatever M is okay, so m dot create tree returns a decision tree just like we asked it to right. So m dot create tree dot. E excess will give us the self ID access inside the decision tree okay, which is set to NP dot. A range range self dot sample size Y is data scientists. Do we care about object-oriented programming, because a lot of the stuff you use is going to require you to implement stuff with oo P? For example, every single pytorch model of any kind is created with olp. It's the only way to create by torch models. Um good news is what you see here is the entirety of what you need to know. So you, this is all you need to know. You need to know to create some in code in it to assign the things to the pasta in it to something call it self and then just stick the word self after it give your methods, okay, and so the nice thing is like now to think. As an AOP programmer is to realize you don't now have to pass around X Y sample size and min leaf to every function that uses them by assigning them to attributes itself they're now available like magic all right. So this is why our peas super handy. If you're, particularly, I started trying to create a decision tree initially without using oot and try to like keep track of like what that decision tree was meant to know about.

It was very difficult. You know where else with our P, you can just say even side. The decision tree - you know self indexes equals this and everything displace okay, okay, that's great! So we're out of time. I think that's that's great timing, because there's an introduction, 200 P, but this week you know next class, I'm going to assume that you can use it right, so you should create some classes. Instantiate some classes, look at their methods and properties. Have them call each other and so forth until you feel comfortable with them, and maybe for those of you doesn't haven't done our P before you can find some other useful resources. You could flop them onto the wiki thread so that other people know what you find useful right. Thanks. Everybody

Lesson 06

Outline

- What makes a good validation set?
- What makes a good test set?
- Random Forest from scratch : setup framework

Video Timelines and Transcript

Note: this lesson has a VERY practical discussion with USF students about the use of Machine Learning in business/corporation, Jeremy shares his experience as a business consultant (McKinsey) and entrepreneur in AI/ML. Deffo not PhD's stuff, too real-life.

1. 00:00:04

- **Review of previous lessons: Random Forests interpretation techniques,**
- **Confidence based on tree variance,**
- **Feature importance,**
- **Removing redundant features,**
- **Partial dependence…**
- **And why do we do Machine Learning, what's the point ?**
- **Looking at PowerPoint 'intro.ppx' in Fastai GitHub: ML applications (horizontal & vertical) in real-life.**
- **Churn (which customer is going to leave) in Telecom: google "jeremy howard data products",**
- **drive-train approach with 'Defined Objective' -> 'Company Levers' -> 'Company Data' -> 'Models'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So we've looked at a lot of different random and random forest interpretation techniques and a question: that's come up a little bit on the forums is like what are these for really like, like how do these help me get a better score on cattle and my answers Kind of been like they don't necessarily, and so I want to talk more about like why do we do machine learning like what's the point and to answer this question, I'm gon na put this PowerPoint in the github repo, so you can have a look. I want to show you something really important, which is examples of how people have used machine learning, mainly in business, because that's where most of you are probably going to end up after this is working for some company. I'm going to show you a plication zuv machine learning which are either based on things that I've been personally involved in myself or know of people who are doing them directly. So these are none of these are going to be like hypotheticals. These are all actual things that people are doing and I'm got direct or secondhand knowledge of I'm going to split them into two groups: horizontal and vertical, so in business. Horizontal means something that you do like across

different kinds of business, whereas vertical means that something that you do within you know, within a business or within a supply chain or within a process. So in other words an example of horizontal applications. Is everything involving marketing.

So like every company, pretty much has to try to sell more products to its customers, and so therefore does marketing, and so each of these boxes are examples of some of the things that people are using machine learning for in marketing. So, let's take an example: all right, let's take Chen okay, so Chen refers to a model which attempts to predict who's going to leave. That's why I've done some churn modeling fairly recently in in telecommunications, and so we're trying to figure out this big cellphone company, which customers are going to leave. That is not of itself that interesting, like building a highly predictive predictive model that says Jeremy. How it is almost certainly going to leave next month is probably not that helpful because, like if I'm almost certainly going to leave next month, there's probably nothing you can do about it, but it's it's too late. What could have cost you too much to keep me right so in order to understand like or why would we do churn modeling I've got a little framework that you might find helpful. So if you, google, for Jeremy Howard data products, I think I've mentioned this thing before there's a paper. You can find designing great data products that I wrote with a couple of colleagues a few years ago and in it I describe my experience of actually turning machine learning models into like stuff. That makes money right and the basic trick.

Is this thing I call the drive train approach, which is, which is these four steps, the starting point to actually turn a machine learning project into something? That's actually useful is to know what am I trying to achieve, and that doesn't mean like I'm trying to achieve a high area under the ROC curve around trying to achieve a large difference between classes. No, it would be I'm trying to sell more books or I'm trying to reduce the number of customers that leave next month or I'm trying to detect lung cancer earlier right. These are things that these are objectives, so the objective is something that absolutely directly is the thing that the the company or the organization actually wants. No company or organization lives in order to create a you know, a more accurate predictive model, but there's some reason right. So that's your objective. Now, that's obviously the most important thing. If you don't know the purpose of what you're modeling for then you can't possibly do a good job of it, and hopefully people are starting to. You know pick that up in out there in the world of data science, but, interestingly, what very few people are talking about, but it's just as important as the next thing, which is levers lever, is a thing that the organization can do to actually drive the objective.

So, let's take the example of churn modeling right: what is a lever that an organization could use to reduce the number of customers that are leaving? They could look, take a closer look at the model and do some of this random forest interpretation and see some of the causes that are causing people to leave and potentially change those issues in the company. Okay. So that's it. That's a data scientist answer, but I want you to go to the next level. What are the things that levers are the things they can do you want to put it past behind you, what are the things that they can do just reach like calling, or else they could call someone and say like? Are you happy anything we could do? Okay yeah, so they could like give them a free pen or something if they, you know, buy 20 bucks worth of product next month. Yep. You are going to do that as well. Okay, so you guys, you guys are the giving out carrots rather than the handing out sticks. You know over in a couple of yeah special all right, so these are levers right, and so, whenever you're working as a data scientist, you know keep coming back and thinking. What are we trying to achieve, we being the organization and how we try to achieve

it? Being like what are the actual things, we can do to make that objective happen. So building a model is never ever a lever, okay, but it could help you with the lever.

So then, the next step is what data does the organization have? That could possibly help them to set that lever to achieve that objective right, and so this is not what data did they give you when you started the project right but like think about it from a first principles, point of view: okay, I'm working for a telecommunications Company they gave me some certain set of data, but I'm sure they must know where their customers live. How many phone calls they made last month? How many times they call customer service whatever like so have a think about like okay, if we're trying to decide like who should we reduce the you know give a special offer to proactively, then we want to figure out like what information do we have? That might help us to identify who's going to react well or badly. To that, perhaps more, interestingly, would be what, if we were doing like a fraud, algorithm right and so we're trying to figure out like who's going to like not pay for the phone that they take out of the store. You know that they once on 12-month payment plan, we never see them again now, in that case, the data we have available. It doesn't matter what's in the database, what matters is what's the data that we can get when the custom is in the shop right, so there's often constraints around the data that we can actually use.

So we need to know what am I trying to achieve? What can I actually? What can this organization actually do specifically to change that outcome and at the point that that decision is being made, what data do they have or could they collect right, and so then, the way I put that all together is with a model, and this is not A model in the sense of a predictive model, but it's a model in the sense of a simulation model. So one of the main examples I gave in this paper is when I spent many years building, which is if an insurance company changes their prices. How does that impact their profitability right and so generally, your simulation model contains a number of predictive models, so I had, for example, a predictive model called an elasticity model. That said for a specific customer if we charge them a specific price for a specific product. What's the probability that they would say yes, both when it's new business and then a year later, what's the probability that they're good new and then there's another predictive model, which is what's the probability that they're going to make a claim and how much is that plan Going to be right, and so like you can combine these models together, then to say all right if we changed our pricing by reducing it by 10 % for everybody through body between 18 and 25, and we can run it through these models that combined together into A simulation than the overall impact on our market share in ten years time is X and our cost is y and our profit is Z and so forth.

Right so in practice, most of the

2. 00:10:01

- **"In practice, you'll care more about the results of your simulation than your predictive model directly ",**
- **Example with Amazon 'not-that-smart'recommendations vs optimization model.**
- **More on Churn and Machine Learning Applications in Business**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Time, you really are going to care more about kind of the results of that simulation than you

do about the predictive model directly, but most people are not doing this effectively at the moment. So, for example, when I go to Amazon right, I read all of Douglas Adams as books, right and so having read all with Douglas Evans's books. The next time I went to Amazon they said, would you like to buy the collected works of Douglas Adams? This is after I had bought every one of his books so like from a machine learning point of view. Some some data scientist had said. Oh people that buy one of Douglas Adams as books often go on to buy the collected works right, but recommending to me that I buy the collected works of Douglas Adams. Isn't smart that and it's actually not smart at a number of levels like not only is unlikely to buy a box set of something of which I have everyone individually, but furthermore, it's not going to change my buying behavior, like I already know about Douglas Adams. I already know I like him so taking up your valuable web space. To tell me hey, maybe you should buy more of the author who you're already familiar with in the port. Lots of times isn't actually going to change my behavior right. So what? If, instead of creating a predictive model, Amazon had built an optimization model that said like that, could simulate and said if we show Jeremy this ad, how likely is he then to go on to buy this book and if I don't show him this ad, how likely Is here to go on to buy this book, and so that's the counterfactual right, the counterfactual is what would have happened otherwise and then you can take the difference and say: okay, what should we recommend him that is going to maximally change his behavior? So maximally result in more books and so you'd probably say like.

Oh he's, never bought me terry pratchet box. He probably doesn't know about terry pratchet, but like lots of people that, like to douglas adams's, did turn out to like terry pratchett. So, let's like introduce him to a new author right, so it's the difference between a predictive model, on the one hand versus an optimization model. On the other hand, so the two tend to go hand in hand right. The optimization model basically is saying well, first of all, we have a simulation model right. The simulation model is saying in a world where we put terry pratchett's book on the front page of amazon for Jeremy Howard. This is what would have happened. He would have bought it with a money, four percent probability right, and so that then tells us with this lever of like what do I put on there on my homepage for Jeremy today, we say: okay, well, the different settings of that lever that put Terry Patchett On the homepage has the highest simulated outcome right and then that's the thing which maximizes our profit from Jeremy's visit to amazon.com today. Okay, so, generally speaking, your predictive models kind of feed in to this simulation model, but you kind of got to think about like how do they all work together. So, for example, let's go back to churn right, so I'd turn out that Jeremy Howard is very likely to leave his cell phone company next month.

What are we gon na do about it? Oh, let's call him right and I can tell you if my cell phone company calls me right now and says just calling to say we love you I'd be like I'm cancelling right now like like. That would be a terrible idea. So again, you would want a simulation model that says like. What's the probability that Jeremy is going to change his behavior as a result of calling him right now right so elite, one of the levers I have is call him. On the other hand, if I like got a piece of mail tomorrow that said like for each month, you stay with us we're going to give you a hundred thousand dollars. Okay, then that's going to definitely change my behavior right so, but then feeding that into the simulation model. It turns out that overall, that would be an unprofitable choice to make. So do you see how this fits in together all right? So so, when we look at something like churn, we want to be thinking like what are the levers. We can pull right, and so what are the kind of models that we could build with what kinds of data to help us pull those levers better to achieve our objectives? And so, when you think about it, that way, you realize that the vast majority of these applications are not

largely about a predictive model at all, they're about interpretation they're about understanding what happens if right.

So if we kind of take the cross-product or not the cross-product or either inter intersection between on the one hand, here are all the levers that we could pull like here or all the things we can do. And then here are all of the features from our random forest feature importance that turn out to be strong drivers of the outcome, and so then the intersection of those is here are the levers. We could pull that actually matter right, because if you can't change the thing that is not very interesting and if it's not actually a significant driver, it's not very interesting right, so we can actually use our random forest feature importance to tell us what can we actually Do to make a difference, and then we can use the partial dependence to actually build this kind of simulation model to say like okay. Well, if we did change that, what would happen? Okay, so you know there are examples, lots and lots of these vertical examples, and so what I want you to kind of think about, as you think, about the machine learning problems you're working on is like. Why does somebody care about this right and like what would a good answer to them look like and how could you you know, how could you actually positively impact this business so, if you're creating like a Keagle, you know try to think about.

From the point of view of the competition organizer like what would they want to know, and how can you give them that information so something like fraud detection? On the other hand, you probably just basically want to know whose fraudulent right, so you probably do just care about the predictive model, but then you do have to think carefully about the data availability here. So it's like, okay, that we need to know who is fraudulent at the point that we're about to deliver them a product right. So it's no point like looking at data. That's available like a month later, for instance, so you've kind of got this. This key issue of thinking about you know the actual operational constraints that you're working under you know lots of interesting applications in human resources, but, like employee churn, it's another kind of churn model, we're finding out that Jeremy Howard sick of lecturing he's going to leave tomorrow. What are you going to do about it? Well, knowing that wouldn't actually be helpful, it'd be too late right, you would actually want a model. That said what kinds of people you know are leaving USF, and it turns out that, like Oh everybody that goes to the downstairs, cafe leaves USF. You know, I guess their food is awful or you know whatever right or everybody, that we're paying less than half a million dollars a year is leaving USF. You know because they can't afford basic housing in San Francisco. So, like you could use your employee churn model, not so much to say like which employees hate us, but why do employees leave right? And so again, it's really the interpretation there that that matters now lead prioritization is a really interesting one right like this is one where a lot of companies.

Yes Dana. Can you pass that over there? You know so I was just wondering, select the Shawn thing or you suggest it. So one being is like being an employee like a 1 million a year or something, but then it sounds like there are two predictors that you need to predict. Put I mean one, each Shawn and one you need to optimize for it like you're profiting, so how does it work yeah exactly so? This is what this like simulation model is all about. So it's a great question so, like you kind of figure out this objective, we're trying to maximize which is like company profitability, you can kind of create like a pretty simple like Excel model or something that says like here's, the revenues and here's the costs and the Cost is equal to them. You know number of people we employ multiplied by their salaries, blah blah blah blah blah right and so inside that kind of Excel model. There are certain cells. There are certain inputs where you're like oh, that thing's kind of stochastic you know or that thing is

kind of uncertain, but we could predict it with a model and so that's kind of what I. What I do, then, is, I then say: okay, we need a predictive model for how likely somebody is to stay if we change their salary, how much? How likely they are to leave. You know with the current salary how likely they are to leave next year if they, if I increase their salary now, why blah? So you could have ruled a bunch of these models and then you can bind them together with simple business logic, and then you can optimize that you can then say.

Okay, if I you know pay, Jeremy, Howard, half a million dollars, that's probably a really good idea. You know, and if I pay him less than you know, it's probably not or whatever like you, can figure out the the overall impact, and so it's it's really shocking to me. How few people do this, but most people in industry measure their models using like a UC or our MSC or whatever, which is never actually what you want.

3. 00:20:30

- **Why is it hard/key to define the problem to solve,**
- **ICYMI: read "Designing great data products" from Jeremy in March 28, 2012 ^!^**
- **Healthcare applications like 'Readmission risk'. Retail applications examples.**
- **There's a lot more than what you read about Facebook or Google applications in Tech media.**
- **Machine Learning in Social Sciences today: not much.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Yes, can you pass it over here? I wanted to stress the point that you made before um. In my experience, a lot of the problem was to define the problem right. So you you are in a company you're talking to somebody that doesn't have like this mentality that you have. They don't know that you have to have x and y and so on. So you have to try to get that out of them. You know what exactly do you want and try to go through a few iterations of understanding what they want, and then you know the data, you know what it is. You know actually what you can measure, which is often know what they want. So you have to kind of get a proxy for what they want and then so a lot of what you do is not that much of like well. Some people do actually just work on really good models for you know, but a lot of people also just work on this kind of. How do you put this? I say you know, classification, regression or some other type of modeling, that's actually kind of the most interesting. I think, and also kind of what's kind of what you have to do well understand the technical model building deeply, but also understand the kind of strategic context deeply, and so this is one way to think about it and, as I say like you know, I actually Think you know there aren't many articles. I wrote in 2012 I'm still recommending, but this one, I think, is still equally valid today, so yeah so like another great example is lead prioritization right.

So, like a lot of companies like every one of these boxes, I'm showing you can generally find a company or many companies whose sole job in life is to build models of that thing right. So there are lots of companies that sell laid prioritization systems, but again, like the question is how would we use that information right? So if it's like, oh our best lead is Jeremy, you know he's a highest probability of buying. Does that mean I should send a salesperson out to Jeremy, or I shouldn't like, if he's highly probable, to buy why I waste my time with him. You know so like again, it's like. He really wants some kind of simulation. That says like what's

the chain, the likely change in Jeremy's behavior, if I send my best salesperson your net out to go and like encourage him to sign okay, so yeah. I think this. This is like. There are many many opportunities for data scientists in the world today to move beyond predictive modeling to actually bringing it all together. You know, and with the kind of stuff that Dena was talking about in the question so as well as these horizontal applications that basically apply to like every company, there's a whole bunch of applications that are specific to like every part of the world right. So if for those of you that end up in health care, some of you will become experts in one or more of these areas like readmission risk.

Okay, so what's the probability that this patient is going to come back to the hospital and readmission is depending on the details of the jurisdiction and so forth, it can be a disaster for hospitals when somebody is readmitted right. So if you find out that this patient has a high probability of readmission, what do you do about it? Well again, the predictive model is helpful of itself right. It rather suggests like we just shouldn't, send them home yet because they're going to come back, but wouldn't it be nice. If we had the tree interpreter - and it said to us the reason that they're at high risk is because we don't have a recent EKG for them and without a recent EKG, we can't have a high confidence about their. You know cardiac health, in which case it wouldn't be like well, let's keep them in the hospital for two weeks, it'll be like, let's give them an EKG okay. So this is. This is interaction between interpretation and predictive accuracy. The predictive models are a really great starting point, but in order to actually like answer these questions, we really need to focus on the interpretability of these models. Yeah, I think so and and more specifically, I'm saying like we just learnt a whole raft of random forest interpretation techniques, and so I kind of just to kind of try to justify like well. Why right and so the reason.

Why is because, actually maybe I'd say most of the time, the interpretation is the thing we care about and, like you can create a chart, you know or a table without machine learning, and indeed that's how most of the world works right. Most managers like build all kinds of tables and charts without any machine learning behind them, but they often make terrible decisions because they don't know the future importance of the objective they're interested in and so the table they create is of things that actually are the least Important things anyway, or they just do a univariate chart rather than a partial dependence plot, so they don't actually realize that the relationship they thought they're looking at is drew entirely to something else right. So you know I'm kind of arguing for data scientists getting like much more deeply involved in in strategy and in trying to use machine learning to really help. You know help a business with all of its objectives. Right now, there's like there, companies like dun hum B, is a huge company that does nothing but retail applications with machine learning and so, like, I believe, there's like a dun Humby product you can buy, which will help you, which will help you figure out like if I put my new store in this location versus lat location.

How much you know how many people are going to shop there or, if I put like you, know my diapers in this part of the shop versus that part of the shop how's that going to impact you know purchasing, behavior or whatever right. So it's I think. It's also good to realize that, like the subset of machine learning applications, you tend to hear about you know in in the tech press or whatever. Is this massively biased, tiny subset of stuff which kind of Google and Facebook do where else the vast majority of stuff that actually makes the world go around? Is you know these kinds of applications that actually help people make things buy things sell things build things so forth, so about create repetition? The way we looked at the tree was we manually check, which feature could cause good was more important for for particular observation, but for businesses they would have a huge amount of data and they they want

this interpretation for a lot of observations. So how do they automate it? I don't think the automation is at all difficult like you just you can run any of these algorithms like looping, through the rows or doing them in parallel. It's all this code. Oh, I miss understand your question. Is it like? They set a threshold that, if some feature is about like four different different people will have different behavior, oh so so yeah, okay, I guess it's good question that the important thing this is a really important issue.

Actually is the vast majority of machine learning models? Don't automate anything they're designed to provide information to humans right so, for example, if you're appointed sales customer service phone operator for an insurance company - and your customer asks you - why is my renewal $ 500 more expensive than last time? Then? Hopefully you know the insurance company has provides in your terminal those little screen that shows the result of the tree interpreter or whatever, then chills so that you can jump there and tell the customer like okay. Well, here's last year, you're in this different zip code, which you know is less, has lower amounts of car theft and this year also you've actually changed your vehicle to more expensive one or whatever right. So it's not so much about thresholds and automation, but about you know making these model outputs available to the decision-makers in an organization whether they be at the top strategic level of like you know, are we going to shut down this whole product or not all the Way to the operational level, look like it that that individual discussion with a customer so like another example, is like aircraft scheduling and gate management like there's lots of companies that do that right and basically, what happens is that the the people you know there are there Are people in at an airport whose job it is to basically tell each aircraft what gate to go to to figure out when to close the doors stuff like that? And so the idea is, you know, you're giving them software, which has the information they need to make good decisions, so the machine learning models end up, embedded in that software to kind of say, like okay, that plane that's currently coming in from Miami there's. A 48 percent chance that it's going to be over five minutes late and if it does, then this is going to be the knock-on impact through the rest of the terminal, for instance, okay.

Well, that's kind of how these things turned a bit together, so there's so many of these right there's lots and lots, and so it's not like. I don't expect you to like remember all these applications, but what I do want you to do is to like spend some time like thinking about them like sit down with one of your friends and like talk about a few examples of like okay, how would we Go about like doing failure, analysis and manufacturing, like you know, who-who would be doing that. Why would be they doing it? What kind of models might they use? What kind of data might they use like start to like practice this and get a sense? So because then this you're like interviewing and then when you're at the workplace and you you know you're talking to managers, you wouldn't be like straightaway able to kind of recognize that the person you're talking to what do they try to achieve. What are the levers that they have to pull right? What if the data they have available to pull those levers to achieve that thing, and therefore how could we build models to help them do that and what kind of predictions would they have to be making right? And so then you can have this really thoughtful. Empathetic conversation with those people and the same like hey, you know, in order to reduce the number of customers that are leaving, you know I guess you're trying to figure out like you know who, should you be providing better pricing to or whatever and so forth? So what I'm noticing like from your beautiful little chart above, is that, like a lot of this, to me at least still seems like.

The primary purpose is like at the at least base level like is predictive power, and so I guess my thing is is like for explanatory problems like a lot of the ones that are people are faced

with, like in social sciences. Is that something machine learning can be used for or is used for, or is that not really the realm that it yeah? It's um, that's a great question and I've had a lot of conversations about this with people in social sciences and currently machine learning is not well applied in, like economics or psychology or whatever on the whole, but I'm Jun convinced it can be, for the exact reasons We're talking about so, if you're, trying to figure out like you're, going try to do some kind of behavioral economics and you're trying to understand like why some people behave differently to other people. You know a random forest with a feature importance. What would be a great way to start or like more interestingly, if you're trying to do some kind of sociology experiment or analysis based on a large social network data set where you have an observational study, you really want to try and pull out all of the Sources of kind of exogenous variables, you know all the stuff that's going on outside, and so, if you use a partial dependence plot with a random forest that happens automatically.

So I actually gave a talk at MIT a couple of years ago for the first conference on digital experimentation, which was really talking about like how do we experiment in you know things like social networks in kind of these digital environments and yeah economists? Economists all do things with, like you know, classic statistical tests but um the group of yeah yeah um, so but anyway, in this case the The Economist's. I talked to were absolutely fascinated by this and they actually asked me to give a a introduction to machine learning session at MIT to these various faculty and graduate folks in the Economics Department and so and some of those folks have gone on to be. You know write some pretty famous books and stuff, and so hopefully it's been useful so like it's definitely early days, but it's a it's. It's a big big opportunity, but as unit says it's you know, there's plenty of skepticism still out there huh. Well. The skepticism comes from unfamiliarity, basically with like this totally different approach so like if you've, if you spent 20 years studying econometrics and somebody comes along and says you know, here's a totally different approach to all the econometrics, all these stuff. That econometricians do you know. Naturally, your first reaction will be like prove it. You know. So let's do enough, but I think it's you know over time. The next generation of people who are growing up with machine learning, some of them will move into the social sciences.

They'll make huge impacts that nobody's ever managed to make before and people will start going wow. You know just like happened in computer vision right when you know computer vision spent a long time of people saying like hey. Maybe you should use deep learning for computer vision and everybody in computer vision is like proven. You know we have decades of work on amazing feature detectors for computer vision and then finally, in 2012 you know Hinton and cadets key came along and said: okay and models like twice as good as yours, and you know, we've only just started on this and everybody Was like okay, that's pretty convincing. Nowadays, every computer vision researcher basically uses the big wedding, so I think that time will come in there in this area too. Okay, I think what we might do, then, is take a break and we're going to come back and talk about these random forest interpretation techniques and do a do a bit of a review. So let's come back at two o'clock, so let's have a go at

4. 00:37:15

- **More on Random Forests interpretation techniques.**
- **Confidence based on tree variance**

Talking about these different random forest interpretation methods, having talked about like why they're important, so, let's now remind ourselves like what they are, so I got ta. Let you folks have a go. So, let's, let's start with confidence based on tree variance, so can one of you tell me one or more of the following things about confidence based on tree variance to do what does it tell us? Why would we be interested in that and how is it calculated? This is going back a ways first, when we looked at even if you're, not sure you only know a little piece of it, give us your piece and we'll build on it together. I think I got a piece of it. It's it's getting. The variance of our predictions from random forests - that's true, that's the! How can you be more specific? What is it the variance of? I think it's remembering correctly. I think it's just the overall prediction: the variance of the predictions of the trees. Yes, so normally the prediction is just the average. This is the variance of the trees, and so it kind of just gives you an idea of how much your prediction is going to vary. So, if maybe even want to minimize variance. Maybe that's your goal for whatever reason that could be: that's not so much the reason. So I, like your calculation description, let's see if somebody else can tell us how you might use that it's okay, if you're not sure, now have a start.

So I remember that we talked about the independence of the trees and so maybe something about if the branch of the trees is higher or lower than no, not so much that um. That's that's that's an interesting question, but it's it's not what we're going to see here. I'm gon na pass it back behind you. So to remind you just to fill in a detail here. What we generally do here is we take us just one row like one observation, often and like find out how confident we are about that like how much variance there are in the trees for that or we can do it, as we did here for different groups For each row, we calculate the standard deviation that we get from the random forest model and then maybe group according to different variables or predictors and see for which particular predictor. The standard deviation is high, then go deep, donnas why it is happening. Maybe it is because a particular category of that variable has very less number of observation. Yeah, that's great, so that that would be one approach is kind of. What we've done here is to say, like is there any groups that have that, where we're very unconfident, something that I think is even more important, would be when you're using this like operationally right, let's say, you're doing a credit, decisioning algorithm, so we're trying to say, Like okay is jeremy, a good risk or a bad risk, should we loan him a million dollars and the random forest says, I think, he's a good risk, but I'm not at all confident, in which case we might say.

Okay, maybe I shouldn't give him a million dollars where else, if we, if, if the Rena first said, I think, he's a good risk, I am very sure of that. Then we're much more comfortable, giving him a million dollars right and I'm a very good risk, so feel free to give me a million dollars right. I checked the random forest before a different notebook, not in the repo so like. This is like it's quite hard for me to give you folks direct experience with this kind of like single observation interpretation stuff, because it's really like the kind of stuff that that you actually need to be putting out to the front line. Do you know what I mean like it's, not something which you can really use so much in a kind of casual context, but it's more like okay, if you're actually putting out some algorithm which is making like big decisions that could cost a lot of money. You probably don't so much care about the average prediction of the random forest, but maybe you actually care about, like the average minus a couple of standard deviations, you know like what's the kind of worst-case prediction and so magic I mentioned it's like. Maybe there's a whole group that we're kind of unconfident about so yeah. So that's

5. 00:42:30

- **Feature importance, and Removing redundant features**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Confidence based on tree variance all right, who wants to have a go with answering feature importance. What is a, why is it interesting? How do we calculate it or any subset thereof? You know, I think it's like that's, basically to find out which, which of those which features are important for your model. So you take each feature and you like randomly sample all the values in the feature, and you see how the predictions are if it's very different, it means that that feature was actually important as if it's fine to take any random values. For that feature, it means that, maybe probably it's not very okay - that was terrific, there's this. That was all exactly right. There was some details that maybe were skimmed over a little bit. I wonder if anybody else wants to jump into like a more detailed description of how it's calculated, because I know this morning, some people were not quite short. Is there anybody who's like not quite sure? Maybe he wants to like have a go or yeah well, it was put it next to there. Let's see how exactly do we calculate feature importance for a particular feature check the validation school if it gets pretty bad for after opening one of the columns. That means that column was important so and as higher importance. I'm not exactly sure how we quantify the feature importance. Okay, great Dana, do you know how we quantify the feature importance? That was a great description or score or some sort exactly yeah.

So let's say, we've got our dependent variable, which is price right and there's a bunch of independent variables, including year made right, and so we basically, we use the whole lot to build a random forest right. And then that gives us our predictions right. And so then we can, let's call this Y right, and so then we can compare that to get. I don't know whatever R squared R MSC, whatever you're interested in right from the model. Now the key thing here is, I don't want to have to retrain my whole random forest, that's kind of slow and boring right, so using the existing random forests. How can I figure out how important year made was right, and so the suggestion was, let's randomly shuffle the whole column right so now that column is totally useless. It's got the same mean same distribution. Everything about it is the same, but there's no connection at all. Between particular people actual year made and what's now in that column, I've randomly shuffled it okay, and so now I put that new version through with the same random forest. So there's no retraining done okay to get some new y hat. I call it Y hat way M right and then I can compare that to my actuals to get like an RSC, ym right and so now I can start to create a little table. So now I can create a little table where I basically got like the original here, our MSC and then I've got with year made scrambled. So this one had an ARMA see of like three.

This one had an ARMA see of like two enclosure scrambling that had a our MSC of like 2.5 right, and so then I just take these differences. So I'd say year made. The importance is one 3-2. Enclosure is 0.53 minus, 2 and 1/2 and so forth. Right, so how much worse did my model get after? I shuffled that variable. Anybody have any questions about that. Can you pass that to Danielle? Please um. I assume you just chose those numbers randomly, but I question I guess is: does it stop? Do all them be ready, I'm not a perfect model to start out with like. Are they welded all the important says seven to one, or is that not no they're? Just I don't honestly, I've never actually looked at what the units are. So I'm not I'm actually not quite sure. Sorry. We can check it out during the week if somebody's interested, how the Vulcans have a look at

the this scikit-learn code and see exactly what those units of measure are cuz. I've never bothered to check, although I don't check, like the units of measure. Specifically, what I do check is the relative importance, and so like here's an example. So, rather than just saying like what are the top ten yesterday, one of the practicum students asked me about a feature importance where they said like. Oh, I think these three are important and I pointed out that the top one was a thousand times more important than the second one right.

So like look at the relative numbers here, and so in that case it's like no don't look at the top three look at the one. That's a thousand times more important and ignore all the rest, and so this is where sometimes the kind of your natural tendency to want to be like precise and careful, you need to override that and be very practical. It's like okay, this thing's a thousand times more important, don't spend any time on anything else right. So then, you can go and talk to the manager of your project and say like okay, this thing's a thousand times more important and then they might say. Oh, that was a mistake. It shouldn't have been in there. We don't actually have that information at the decision. Time or you know, but for whatever reason we can't actually use that variable, and so then you could remove it and and have a look or they might say gosh. I had no idea that, like that was more by far more important than everything else put together. So, let's forget this random virus thing and just focus on like understanding how we can better collect that one variable and better use that one variable. So that's like something which comes up quite a lot and actually another place that came up just yesterday again. Another practicum student asked me: hey I'm doing this medical diagnostics project and my r-squared is 0.95 for a disease which I was told, is very hard to diagnose. You know, is this random forrester genius or is something going wrong, and I said like remember the second thing you do after you build a random forest is to do feature importance, so do feature importance, and what you'll probably find is that the top column is something That shouldn't be there, and so that's what happened.

He came back to me half an hour later, he said yeah. I did the feature importance. You were right. The top column was basically a something that was another encoding of the dependent variable. I've removed it, and now my a squared is negative 0.1. So that's an improvement. Okay. The other thing I like to look at is this chat. Right is to basically say, like you know, where the kind of things flatten off in terms of like which ones should I be really focusing on. So that's the most important one right, and so when I did credit scoring in telecommunications, I found there were nine variables that basically predicted very accurately. Who was it going to end up paying for their phone and who wasn't and like, apart from ending up with a model that saved them three billion dollars a year in in fraud and credit costs? It also let them basically rejigger their their process. So they focused on collecting those nine variables, much better, all right who

6. 00:50:45

- **Partial dependence (or dependance)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Wants to do partial dependence. This is an interesting one, very important, but you know somebody's kind of tricky to think about yeah. Please do not always necessarily like a relationship between the strictly the dependent variable in this independent variable. That necessarily like is showing importance, but rather than an interaction between two variables

that are working together, right, yeah, oh, that's, weird, yeah, and so for this example. What we found was that it's not necessarily your maid or when the sale was elapsed, but it's actually the age of the model, and so it's that is easier to just like to tell a liked company. Well, obviously, you're younger models are gon na sell for more and it's less about when the year was made yeah. So, let's come back to how we calculate this in a moment, but the first thing to realize is that the vast majority of the time you know post your course here when somebody shows you a chart, it'll be like a univariate chat, that'll just like grab the Data from the database and they'll plot X against Y and then managers have a tendency to want to like make a decision. All right so be like. Oh there's this like drop off here. So we should like stop dealing in equipment made between 1990 and 1995 or whatever right, and this is like a big problem because, like real world data, has lots of these interactions going on so, like you know, maybe there was a recession going on around the time That those things are being sold or maybe around that time people were buying more of a different type of equipment or whatever right so generally.

What we actually want to know is all other things being equal. What's the relationship between ear made and sale price right because, like if you think about the drivetrain approach idea of like the levers, you really want a model that says if I change this lever, how will it change my objective, okay and it's by pulling them apart? Using partial dependence that you can say: okay, actually, this is the relationship between year made and sale price, all other things being equal right. So how do we calculate that? For the variable you made, for example, you're gon na train? You keep every other car variable constant and then you're gon na pass every single value of the year mate and then train the model after that. So for every model you might have their light blue or the values of it and the median is gon na. Be the yellow line? Okay, so let's try and draw that so bye leave everything else constant. What she means is leave them at whatever they are in the data set so just like when we did feature importance right, we're going to leave the rest of the data set as it is, and we're going to do. Partial dependence plot for year made all right. So we've got all of these other rows of data that we'll just leave as they are, and so, instead of randomly shuffling em8. Instead, what we're going to do is replace every single value with exactly the same thing: 1960, okay and just like before.

We now pass that through our existing random forests, which we have not retrained or changed in any way to get back out a set of predictions, why 1960, okay and so then we can plot that on a chart, yeah I'd against partial dependence, 1960 here, okay, now We can do it for 1960, one two, three, four five and so forth right, and so we can do that for four on average for all of them, or we could do it just for one of them right and so when we do it for just one Of them and we change its year may to pass that single thing through our model. That gives us one of these blue lines. That's a each one of these blue lines is a single row as we change its year made from 1960 up to 2008, and so then we can just take the median of all of those blue lines to say you know on average, what's the relationship between year Made and price all other things being equal, so why is it that? Why is it that this works? Why is it that this process tells us the relationship between year made and price, all other things being equal? Well, maybe it's good to think about like a really simplified approach, a really simplified approach would say: what's the average auction you know, what's the average sale date, what's the most common type of machine we sell, which location do we mainly mostly sell things and like we Could come up with a single row that represents the the average option and then we could say: okay, let's run that row through the random forest, but replace its year made with 1960 and then do it again with 1961 and then do it again with 1962 and We could like plot, you know those on our little chart right and that would give us a version of the

relationship between year made and sale price, all other things being equal right. But what if like tractors, looked like that and backhoe loaders looked like that right then taking the average one would hide the fact that there are these totally different relationships right.

So instead we basically say: okay, our data tells us what kinds of things we tend to sell and who we tend to sell them and when we tend to sell them. So let's use that right. So then we actually find out like for every blue line like here are actual examples of these relationships right and so then, what we can do is as well as flooding the median is. We can do a cluster analysis to find out like a few different shapes right, and so we may find in this case they all look like pretty much the different versions of the same thing with different slopes. So my main takeaway from this would be that the relationship between sale, price and year is basically a straight line. Right and remember, this was log of sale price right, so this is actually showing us an exponential, and so this is where I would then like bring in the domain expertise which is like okay, things depreciate over time by a constant ratio. So therefore, I would expect older stuff year made to have this exponential shape, so this is where, like I said of mentioned, like the very start of my machine learning project, I generally try to avoid as using as much domain expertise as I can and let the Data do the talking all right so, like one of the questions I got this morning was like if there's like a sale, I'd be a model ID. I should throw those away right because they're just IDs, no, don't assume anything about your data right leave them in and if they turn out to be super important predictors, you want to find out.

You know. Why is that? Okay? But then now I'm at the other end of my project right, I've done my feature importance. I've pulled out the stuff which is like you know from that dendogram. You know the kind of redundant features, I'm looking at the partial dependence and now I'm thinking like okay. Is this shaped what I expected? Okay, so even better before you plot this. First of all, think what shape would I expect this to be because it's always easy to justify to yourself after the fact. Oh, I knew it would look like this right, so what straight you expect and then is it that shape so in this case I'd be like yeah. This is this is what I would expect. Okay, where else this is definitely not what I'd expect. So the partial dependence plot has really pulled out the underlying truth. Okay, does anybody have any questions about like why we use partial dependence or how we calculate it? It's got the. Are you better, don't say you have a few thousand say twenty features. Everything are important. Are you gon na measure the partial dependence for every single one of them? If there are twenty features that are important, then I will do the partial dependence for all of them. We're important means like it's a lever. I can actually pull it's like. The magnitude of its size is like not much smaller than the other nineteen. Like you know, based on all of these things, it's like yeah, it's a feature I ought to care about. Then I will want to know how it's related.

It's pretty unusual to have that many features that are important both operationally and from a modeling point of view. In my experience, so how do you define internationally? So important means it's it's a lever, so it's something I can change and it's like you know, kind of at the spiky end of this tail or you know it. Maybe it's not a lever directly like maybe it's like zip code, and I can't actually tell my customers where to live, but I could like focus my new marketing attention on a different, zip code. You know, would it make sense to do pairwise shuffling for every combination of two features and hold everything else constant, like in future importance to see interactions and compare scores, so you wouldn't do that so much for partial dependence. I think your question is really getting to the question of. Could we do that for feature importance all right, so I think interaction. Feature importance is a very important and interesting question, but doing doing it by randomly shuffling every pair of columns. You know if you've got a hundred columns, sounds computationally intensive,

possibly infeasible. So what I'm going to do is after we talk about tree interpreter I'll, talk about interesting, but largely unexplored approach that will probably work. Okay, who wants to do

7. 01:02:45

- **Tree interpreter (and a great example of effective technical communications by a student)**
- **Using Excel waterfall chart from Chris**
- **Using 'hub.github.com', a command-line wrapper for git that makes you better at GitHub.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Tree interpreter all right over here Prince. Can you pass that over here to principal? I was thinking this to be more like feature importance, but teacher importance is for complete random forest model, and this tree interpreter is for feature importance for particular observation. So if that, let's say it's about hospital readmission, so if a patient a one is, it is going to be readmitted to a hospital which featured for that particular patient is going to impact. And how can we change that, and it is calculated starting from the prediction of mean then seeing how each feature is changing the behavior of that particular patient. I'm smiling, because that was one of the best examples of technical communication. I've heard in a long time. So it's really good to think about like what. Why was that effective right? So what Prince did there was he used as specific an example as possible right so if humans are much less good at understanding, abstractions right? So if you kind of say, oh, it takes some kind of feature and then there's an observation in that feature river. You know where's like know it's it's the hospital readmission, okay, and so we take a specific example. The other thing he did was very effective was to kind of take an analogy to something we already understand. So we already understand the idea of feature importance across all of the rows in a data set. So now we're going to do it for a single row.

Okay, so, like you know, one of the things I was really hoping we would learn from from this experience is how to become effective technical communicators. So you know that was a really great role model from Prince of like using all the tricks we have at our disposal for effective technical communication. So hopefully you found that useful explanation. I don't have a hell of a lot to add to that other than to show you. You know what that looks like so with the tree interpreter. We peaked out a row, okay and so remember when we talked about the the confidence intervals. At the very start, the confidence based on tree variance - we mainly said - like you, probably mainly use that for a Rome, so this would also be for our crow. So it's like okay, why is this patient likely to be readmitted? Okay, so here is all of the information we have about that patient or in this case this option. Now, why is this auction so expensive? So then we call tree interpreter dot predict and we get back the prediction of the price right. The bias which is the root of the tree, so this is just the average price for everybody, so this is always going to be the same and then the contributions, which is how important is each of these. Each of these things right and so the way we calculated that so the way we calculated, that was to say, okay at the very start, the average price was ten right and then we split on enclosure right and for those with dissin closure.

The average was nine point five and then we split on year made - I don't know less than 1990

and for those with that year made the average price was nine point, seven right, and then we split on the number of hours on the meter and for you Know with this branch we got nine point four all right, and so we then have a particular option which we we pass it through the tree, and it just so happens that it takes this path all right. So one row can only have one path through the tree right, and so we ended up at this point. Okay, so then we can create a little table right and so as we go through, we start at the top and we start with 10 right. That's that bias - and we said enclosure resulted in a change from 10 to 9 and 1/2 minus 0.5 yeah man changed it from nine point: five to nine point, seven so plus 0.2 right and then meter changed it from nine point: seven down to nine point: four, Which is minus 0.3 and then, if we add all that together can minus a half is nine and a half plus 0.2 is nine point. Seven minus point: three is nine point four low and behold that's that number, which takes us to our Excel spreadsheet. Where's Chris, who did our waterfall there? You are all right so last week we had to use Excel for this because there isn't a good Python library for doing waterfall charts, and so we saw we got our starting point.

This is the bias and then we had each of our contributions and we ended up with our total. The road is now a better place, because Chris has created a Python waterfall chart module for us and put it on pip, so never again where we have to use Excel for this, and I wanted to point out that, like waterfall, charts have been very important in Business communications, at least as long as I've been in business. So that's about 25 years Python. Is you know what couple of decades old a little bit less yeah, maybe a couple of decades old. But you know, despite that, no one in the Python world ever got to the point where they actually thought. You know I'm gon na make a waterfall chart, so they didn't exist until two days ago, which is to say, like the world, is full of staff which ought to exist in, doesn't and doesn't necessarily take a hell, a lot of time to build Chris. How long did it take you to build the first Python waterfall chart? Well, there was a you know, a gist of it: yeah yeah yeah about eight hours. Okay, so you know a hefty TYIN amount, but not unreasonable and now forevermore people when they want the place and waterfall chart, will end up at Chris's, github repo and hopefully find lots of other USF contributors who have made it even better so for in order for You to help improve Chris's plaything waterfall. You need to know how to do that right and so you're going to need to submit a pull request.

Life becomes very easy for submitting pull requests if you use something called hub. So if you go to github, slash hub, that will send you over here and what they suggest you do is that you alias get to hub, because it turns out that hub actually is earth strict superset if get, but what it lets you do is you can Go get fork, get push, get pull request and you've now sent Chris a pull request now without hub. This is actually a pain and requires like own to the website and filling informants and stuff right. So this gives you no reason not to do pull requests, and I mention this because, like when you're interviewing for a job or whatever, I can promise you that the person you're talking to will check your github. And if they see you have a history of submitting thoughtful pull requests that are accepted to interesting libraries. That looks great right. It looks great because it shows you're somebody who actually contributes. It also shows that, if they're being accepted that you know how to create code that fits with people's coding standards has appropriate documentation, passes their tests and coverage and so forth right. So when people look at you - and they say - oh here's - somebody with a history of successfully contributing accepted, pull requests to open-source libraries.

That's a great part of your portfolio, okay and you can specifically refer to it right so either I'm the person who built Python waterfall here is my repo or you know, I'm the person who

contributed currency, number formatting to Python waterfall, here's my pull request. You know that anytime, you see something that doesn't work right in any open source software you use is not a problem, it's a great opportunity because you can fix it and send in the pull requests so yeah give it a go. It actually feels great. The first time you have a pull request accepted and, of course, one big opportunity is the faster, a library and thank you who it was the person here, the person who added all the docs to fast a I structured in the other class okay. So, thanks to one of our students, we now have docstrings for most of the FASTA. I dot structured library again came by a pull request. So thank you. Okay, does anybody have any questions about how to calculate any of these random forest interpretation methods or why we might want to use any of these random first interpretation methods towards the end of the week you're going to need to be able to build all of these Yourself from scratch over there can you pass that test? That's right! Just looking at the tree interpreter, I noticed that some of the the values are in a ends. How I got I get my you keep them in the tree, but how can an NA and have a future importance? Okay, let me pass it back to you.

Why not so, in other words, how is ni n handled in pandas and therefore in the tree, such as some default value, everybody remember how pandas these are. Notices are all in categorical variables. How does pandas handle an NA ends in categorical variables and how does fast? Ai deal with them can somebody pass it to the person who's talking negative one through yeah pandas sets into negative one category code, and do you have to remember what we then do? Doesn't matter really, we add one to all of the category codes, so it ends up being zero, so in other words, we have a category with remember by the time it hits the random forest is just a number, and it's just it's just the number, zero right And we map it back to the descriptions back here. So the question really is: why shouldn't the random first be able to split on zero? It's just it's just another number, so it could be na n high, medium or low zero. One two three four, and so you know missing values are one of these things that are generally taught really badly. Like often people get taught like here are some ways to remove columns with missing values or remove rows with missing values or to replace missing values. That's like never what we want, because missingness is very, very, very often interesting, and so we actually learnt that from our future importance that cupola system n, a n is like one of the most important features, and so for some reason. Well, I could.

I could guess right: coupla system n am presumably means. This is a kind of industrial equipment that doesn't have a coupler system. Now I don't know what kind that is, but apparently it's more it's a more expensive kind. That makes sense. Yeah. Ok, ok! So I did this competition for university grant research success where, by far the most important predictors were whether or not some of the fields were a null, and it turned out that this was data leakage that these fields only got filled in most of the time. After a research grant was accepted right, so you know it allowed me to win that cowgirl competition, but didn't actually help the university very much okay, great

8. 01:16:15

- **Extrapolation, with a 20 mins session of live coding by Jeremy**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So, let's talk about um extrapolation and I am going to do something risky and dangerous, which is we're going to do some live coding and the reason we're going to do some live

coding is I want to explore extrapolation together with you, and I kind of also Want to get kind of help give you a feel of. You know like how you might go about, like writing - writing code quickly in this notebook environment right, and this is the kind of stuff that you're going to need to be able to do. You know in the real world and in the exam, is kind of quickly create the kind of code that we're going to talk about. So I really like creating synthetic datasets anytime, I'm trying to like investigate the behavior of something, because if I have a synthetic data set, I know how it should behave which reminds me before we do this. I promised that we would talk about interaction importance, and I just about forgot, trie interpreter - tells us the contributions for our particular row. Based on the difference in the tree. We could calculate that for every road in our data set and add them up right, and that would tell us that would tell us feature importance and would tell us feature importance in a different way. Right. One way of doing feature importance is by shuffling the columns. One at a time, another way is by doing tree interpreter for every row and adding them up. Neither is right more right than the others, they're, actually both quite widely used.

So this is kind of type 1 and type 2 feature importance. So we could try to expand this a little bit. Do not just single variable feature importance, but interaction feature importance. Now, here's the thing what I'm going to describe is very easy to describe. It was described by Bremen right back when random forests were first invented, and it is part of the commercial software product from Salford systems who have the trademark on random forests. But it is not part of any open-source library, I'm aware of, and I've never seen an academic paper that actually studies it closely. So what I'm going to describe here is a huge opportunity, but it's also like there's lots and lots of details that kind of need to be fleshed out, but here's the basic idea this particular this particular difference here right is not just because of year made, but Because of a combination of year made and enclosure right, the fact that this is 9.7 is because enclosure was in this branch and year made was in this branch. So, in other words, we could say the contribution of enclosure interacted with year made is minus 0.3 yeah, and so what about that difference? Well, that's an interaction of year made and hours on the meter, so you made interacted with, am using star here not to mean times but to mean interacted with it's a it's a kind of a common way of doing things like ours formulas.

Do it this way, as well here made by interacted with meter, has a import has a importance, sorry, a contribution of minus 0.1. Perhaps we could also say from here to here that this also shows an interaction between meter and enclosure like with one thing in between them, so maybe we could say meter by enclosure equals and then what should it be? You know should it be 0.6? I sorry minus 0.6. I mean some ways that kinda seems unfair because we're also including like the impact of a vm aid right. So maybe it should be. Maybe it should be minus 0.6. You know - maybe we should add back this point to right, and these are like details that I actually don't know the answer to right like how should we best kind of assign a contribution to each pair of variables in this path right, but but clearly conceptually. We can write the pairs of variables in that path. All represent interactions, all right. Yes, Chris can you? Could you pass that to Christmas? Why don't you first have to be next to each other in the tree. I I mean I'm not gon na say it's the wrong approach. I I don't think it's the right approach, though, because it feels like this path here. Mayda and enclosure are interacting, so it seems like not recognizing that contribution is throwing away information, but I'm not sure you know.

I had one of my staff at cattle actually do some R & amp D on this a few years ago and actually found you know - and I wasn't close enough to know how they dealt with these details, but they got it working pretty well, but unfortunately it Never saw the light of day is a software product, but like this is something which you know. Maybe a group of you could get

together and build. You know I mean, do some googling to check, but I really don't think that there are any interaction feature importance. Parts of any open source library can you cross the flip. Wouldn't this exclude interactions, though, between variables that don't matter until they interact so say your road never chooses to split down that path, but that variable interacting with another one becomes your most important split. I don't think that happens right, because if this, if there's an interaction, that's important only because it's an interaction and not in a univariate basis, it will appear. Sometimes they cut, assuming that you set max features to less than one, and so therefore it will appear in in some file what is meant by indirection or is it multiplication ratio addition? Interaction means appears. Branches appears on the same path through a tree like an interaction, in this case the tree there's an interaction between enclosure and ear made because we branched on enclosure, and then we branch on your mode. So to get to here.

We have to have some specific value of enclosure and some specific value of you made sorry. I just membranes kind of working on this right now. What if? What? If you went down the middle Leafs between the two things, you're trying to observe - and you could just sort of norm - and you would also take into account what the final measure is, so I mean if we extend the tree downwards - you'd have many measures both of Like the two things you're trying to look at, and also the in between steps, there seems to be a way to like average information out in between them. There could be so I think what we should do is talk about this on the forum. I think this is fascinating and I hope we build something great, but I need to do my live coding. So, let's yeah, that's a great discussion, thinking about it and news of experiments and so to experiment with that. You almost certainly want to create a synthetic data set first right. It's like y equals X, 1 plus X, 2 plus X, 1 times X, 2 or something you know like something where you know. There's this interaction effect and there isn't that interaction effect. And then you want to make sure that the feature importance you get at the end is what you expected right, and so probably the first step would be to do single variable feature importance using the tree interpreter, style approach, and one nice thing about this is like It's it doesn't really matter how much data you have like all you have to do to calculate feature.

Importance is just like slide through the tree right, so you should be able to write in a way. That's actually pretty fast, and so even writing. It in pure Python might be fast enough, depending on your tree size. Okay, so we're going to talk about extrapolation, and so the first thing I want to do is create a synthetic data set that has a simple linear relationship. We're going to pretend it's like a time series right, so we need to basically create some X values. So the easiest way to kind of create some synthetic data of this type is to use linspace, which just creates some evenly spaced. Some evenly spaced data right between start and stop with by default 50 observations. So if we just do that right there, it is okay and so then we're going to create a dependent variable. And so, let's assume there's just a linear relationship between x and y and, let's add a little bit of randomness to it right so uniform random between low and high. So we could like add somewhere between like minus 0.2 and 0.2, say okay, and so the next thing we need is is a shape right, which is basically how what dimensions do you want this, this randomness, these random numbers to be, and obviously we want them to Be the same shape as X's shape, so we can just say X, dot shape, okay, so, in other words, that's xscape.

Remember when you see something in parentheses with a comma, that's at Apple with just one thing in it: okay, so this is of shape 15 and so we've added 50 random numbers, and so now we could plot those okay, so shift-tab, x, comma y, all right! So there's no data, okay, so

like when you were both working as a data scientist or for doing your exams in this course. You need to be able to like quickly whip up a data set like that, throw it up in our plot without thinking too much. Okay and like, as you can see, you don't have to really remember much if anything you just have to know how to like hit shift. I have to check the names of parameters, and you know everything in the exam will be open up and broke up and internet, so you can always like google her something to try and find linspace if you forgot what it's called all right. So, let's assume that's out data all right, and so we're now going to build a random first model and what I want to do is build a random first model that kind of acts as if this is a time series. So I'm going to take this as a training set right. I'm going to take of this as our validation or test set just like we did in you know, groceries or bulldozers or whatever okay. So we can use exactly the same kind of code that we used in split bells right.

So we can basically say X, train comma X, tau equals x up to 40 comma X from 40 okay, so that just fits it into the first audio since the last 10 right, and so we can do the same thing for Y and there we go okay. So the next thing to do is we want to create a random forest, okay and fit it, and that's going to require X's and Y's all right. Now, that's actually going to give an error and the reason why is that it expects X to be a matrix? Not a vector because it expects X to have a number of columns of data right, so it's important to know that a matrix with one column is not the same thing as a vector all right. So if I try to run this right, expect a 2d array got 1d array instead, so we need to convert our 2d array into a 1d array. So remember I said X, dot shape is 50 comma right, so X has one axis. So here's important to measure X is rank is 1. The rank of a variable is equal to the length of its shape. How many axes does it have so a vector we can think of as an array of Rank 1 matrix as an array of Rank 2? I very rarely used words like vector and matrix because, like they're kind of meaningless, specific examples of something more general, which is they're all n, dimensional, tensors, right or n, dimensional, arrays, ok, so an n dimensional array, we can say it's a tensor of Rank. They basically mean kind of the same thing: physicists get crazy when you say that, because to a physicist, a tensor has quite a specific meaning, but in machine learning we generally use it in the same way.

Okay. So how do we turn an array, a one-dimensional array, into a two-dimensional array? There's a couple of ways: we can do it, but basically we slice it right. So colon means give me everything in that axis: right, colon, comma, none means give me everything in the first axis, which is the only axis we have, and then none is a special indexer, which means add a unit axis here. So let me show you that is a sort of shape: 50, comma one. So it's a prank. It has two axes. One of them is a very boring access right. It's a length, one access. So, let's move this over here, there's 1, comma, 50, ok and then to remind you, the original is just 50 right, so you can see I can put none as a special indexer to introduce a new unit axis there. Ok, so this thing has one row in 50 columns. This thing has 50 rows of one column. So that's what we want right. We want 50 rows and one column. This kind of playing around with ranks and dimensions is going to become increasingly important in this course and in the deep learning course right. So I spend a lot of time slicing with non slicing, with other things, try to create three dimensional, four dimensional chances and so forth. I'll show you a trick I'll show you two tricks. The first is you never ever need to write karma, it's always assumed.

So if I delete that this is exactly the same thing, okay and you'll see that in code all the time, so you need to recognize it. The second trick is this is adding an axis in the second dimension right or I guess the index one dimension. What, if I always want to put it in the last dimension right and like often our tensors change dimensions without us? Looking because, like you, went from a one channel image to a three channel image, or you went from a single image to a mini batch of images like suddenly, you get new dimensions appearing so to make things

general, I would say this dot: dot, dot, dot, dot, Dot means as many dimensions as you need to fill this up. Okay, and so in this case, it's exactly the same thing, but I would always try to write it that way, because it means it's going to continue to work. As I get you know, higher dimensional tenses alright, so in this case I want 50 rows in one column, so I'll call that say X, okay, so let's now use that here and so this is now a 2d array, and so I can create my random forest. Okay, so then I could plot that - and this is where you're going to have to turn your brains on, because the folks this morning got this very quickly, which was super impressive. I'm going to plot white rain against MDOT predict next rain. Okay, before I hit go, what is this going to look like yeah? It should basically be the same right. A predictions hopefully are the same as the actuals, so this should fall on a line, but there's some randomness, so it won't. Quite. I should have used scatter plot, okay, all right, so that's cool right.

That was the easy one. Let's now do the hard one, the fun one. What's that going to look like you? Okay, so I'm gon na say no, but nice try. You know it's like hey we're extrapolating to the to the validation. That's what I'd like it to look like, but that's not what it is going to look like think about what trees do and think about. Think about the fact that we have a validation set here and a training set here so think about a forest is just a bunch of trees. Well, the first tree is going to okay, Melissa is going to have a go. Can you pass that to Melissa? Well, let's start grouping yeah, that's what I mean: that's what it does okay, but you know, let's think about how it groups the dots so yeah, I'm guessing, since all the new data is actually outside of the original scope, it's all going to be. Basically the same. It's like one huge group, yeah right so like we make it like, forget the forest. Let's create one tree right, so we're probably going to split somewhere around here first and then we're kind of probably split somewhere around here and then we're going to split somewhere around here and somewhere around here right, and so our final split is here right. So our prediction: when we say okay, let's take this one, and so it's going to put that through the forest right and end up predicting this average. It can't predict you anything higher than that because there is nothing higher than that to average right.

So this is really important to realize if a random forest is not magic right, it's just returning the average of nearby observations where nearby is kind of in this like tree space. So, let's run it, let's see if Tim's right but holy, that's awful right and like if you don't know how, when firsts works - and this is going to totally screw right, if you think that it's actually going to be able to extrapolate to any kind of data, It hasn't seen before, like particularly like future time periods. It's just not like it. Just can't it's just averaging stuff. It's already seen. That's all it can do okay, so we're going to be talking about like how to avoid this problem. We talked a little bit in the last lesson about trying to avoid it by just like in avoiding unnecessary time, dependent variables where we can write, but in the end, if you really have a time series that looks like this, we actually have to deal with a Problem all right, so one way we could deal with the problem would be used like a neural net right use, something that actually has a function or shape that can actually like fit something that can she fit something like this right and so then it will extrapolate Nicely another approach would be to use all the time series techniques you guys are learning about in the morning class to fit some kind of time, series right and then D trend it right and so then you'll end up with D, trended dots and then use the Random chorus to predict those right and that's particularly cool right, because, if you're like imagine that your random forest was actually trying to predict data that, like I don't know, maybe it was two different states and so the blue ones. You know that down here and the red ones are up here right now.

If you try to use a random forest, it's going to do a pretty crappy job because, like time is

going to see much more important, so it's basically still gon na. Like split like this, and it's going to split like this and then finally, once it kind of gets down to this piece, it'll be like oh okay. Now I can see the difference between the states right. So, in other words like when you've got this big timepiece going on you're, not gon na see the other relationships in the random forest until you've dealt and kill every tree deals with time. So one way to fix this would be with a gradient, boosting machine GBM right now. What a GBM does is. It creates a little tree right and runs everything through that first little tree, which could be like the time tree, and then it calculates the residuals and then the next little tree just predicts the residuals, so it'd be kind of like D trending it right. So GPM has handle this GBM still can't extrapolate to the future, but at least they can deal with time-dependent data more conveniently right. So we're going to be talking about this quite a lot more over the next couple of weeks, right and in the end that a solution is going to be just used neural nets right, but for now you know using it some kind of time series analysis, D Trend it and then use a random forest on that, isn't a bad technique at all and if you're playing around something like the Ecuador groceries competition.

That would be a really good thing to to fiddle around with alright see you next time.

Lesson 07

Outline

- Motivations for data science
- Thinking about the business implications
- Tell the story
- Review of Confidence in Tree Prediction Variance, Feature importance, Partial Dependence

Video Timelines and Transcript

1. 00:00:01

- **Review of Random Forest previous lessons,**
- **Lots of historical/theoritical techniques in ML that we don't use anymore (like SVM)**
- **Use of ML in Industry vs Academia, Decision-Trees Ensemble**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Welcome back we're going to be talking today about random forests, we're going to finish building our own random forest from scratch. But before we do, I wanted to tackle a few things that have come up during the week. A few questions that I've had and I want to start with kind of the position of random forests in general, so we spent about half of his course doing random forests and then, after today, the second half of this course will be neural networks broadly defined. This is because these these to represent, like the tea, the two key classes of techniques, which cover nearly everything that you're likely to need to do random forests belong to the class of techniques of decision tree ensembles, along with gradient, boosting machines being the other key type And some variants, like extremely randomized trees, they have the benefit that they're highly interpretive all scalable flexible work well for most kinds of data. They have the downside that they don't extrapolate at all to like data. That's outside the range that you've seen as we looked at at the end of last week's session, but you know they're, there they're a great starting point, and so I think you know there's a huge catalogue of machine learning tools out there and I got lot of Courses and books, don't attempt to kind of curate that down and say like for these kinds of problems, use this for these kinds of problems.

Is that finished you know, but they're rather like here's, a description of 100 different algorithms and you just don't need them. You know like I, don't see why you would ever use in support, vector machine today, for instance like not know, no reason at all. I could think of doing that. People love studying them in the 90s because they are like very theoretically elegant and like you can really write a lot of math about support, vector machines and people did. But you know in practice: I don't see them as having any place. There's like a lot of

techniques that you could include in an exhaustive list of every way that people adopt machine learning problems. But I would rather tell you like how to actually solve machine learning problems in practice. I think they, you know, we've we've about to finish today. The first class, which is you know, one type of decision tree ensembles in in part to you net, will tell you about the other key type there being gradient, boosting and we're about to launch next lesson into neural nets, which includes all kinds of GLM Ridge regression Elastic net lasso, logistic regression, etc, or all variants of neural nets. You know, interestingly, leo breiman, who created random forests, did so very late in his life and unfortunately passed away, not many years later, so, partly because of that very little has been written about them in the academic literature, partly because SVM's were just taken over at that Point you know other people didn't look at them and also like just because they're like quite hard to grasp at a theoretical level like analyze them, theoretically, it's quite hard to write conference papers about them or academic papers about them.

So there hasn't been that much written about them, but there's been a real resurgence or not resurgence. A new wave in recent years of empirical machine learning like what actually works, kegels been part of that, but also just in part of it, has just been like companies using machine learning to make loads of money like Amazon and Google, and so nowadays a lot of People are writing about decision tree ensembles in creating better software for decision tree ensembles like like GBM and x3 boost and Ranger, 4r and scikit-learn and so forth. But a lot of this is being done in industry rather than academia, but you know it's. It's encouraging to see. There's certainly more work being done in deep learning than in decision tree ensembles, particularly in in academia, but but there's a lot of progress being made in both. You know if you look at like of the packages being used today for decision tree ensembles like all the best ones, the top five or six. I don't know that any of them really existed five years ago. You know maybe other than like SK, learn or even three years ago. So yet so that's that's been good, but I think there's a lot of work still to be done. We talked about, for example, figuring out what interactions are the most important last week, and some of you pointed out in the forum's that actually there is such a project already for a gradient, boosting machine which is great, but it doesn't seem that there's anything like that.

Yet for random forests - and you know random forests - do have a nice benefit over gbms that they're kind of harder to screw up.

2. 00:05:30

- **How big the Validation Set needs to be ? How much the accuracy of your model matters ?**
- **Demo with Excel, T-distribution and n>22 observations in every class**
- **Standard Deviation : n*p*(1-p), Standard Error (stdev mean): stdev/sqrt(n)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

You know and easier to scale, so hopefully that's something that you know this community might help fix. Another question I had during the week was about the size of your validations that how big should it be so like to answer this question about? How big does your validation set need to be? You first need to answer the question: how how accurate do I need help recite to know the accuracy of this algorithm right so like if the validation set that you have is saying like this is 70 % accurate and if somebody said well, is it 75 % Or 65 % or 70 % - and the

answer was, I don't know anything in that range is close enough like that would be one answer where else, if it's like is that 70 percent or 70 point? Oh one percent or sixty nine point, nine nine percent. Like then, that's something else again right, so you need to kind of start out by saying like how how accurate do I need this so like, for example, in the deep learning course, we've been looking at dogs versus cats images and the models that we're looking at Had about a ninety nine point, four, ninety nine point: five percent accuracy on the validation set and a validation set size was 2000. Okay. In fact, let's do this in Excel. That'll, be a bit easier, so our validation set size was 2000 and was 99.4 % right. So the number of incorrect is something around one accuracy times, and so we were getting about 12 wrong right and the number of cats we had is 1/2, and so the number of wrong cats is about 6. Ok, so then, like we, we run a new model and we find instead that the accuracy has gone to 99.2 % right and then it's like.

Okay, is this less good at finding cats? That's like! Well, it got 2 more cats wrong. So it's like, probably not right so, but then it's like well does this matter. There's ninety nine point: four versus ninety nine point, two matter and if this was like it wasn't about cats and dogs, but it was about finding fraud right then, the difference between a point, six percent error rate and a point - eight percent error rate - is like twenty Five percent of your cost of fraud so like that can be huge like it's really interesting like when imagenet came out earlier this year, the new competition results came out and the accuracy had gone down from three percent, so the error went down from three percent to Two percent and I saw a lot of people on the internet like famous machine learning, researchers being like yeah, some Chinese guys got it better from like 97 % to one ninety eight percent. It's like, statistically, not even significant, who cares kind of a thing, but actually I thought like holy crap, this Chinese team just blew away the state-of-the-art. An image recognition like the old one was fifty percent less accurate than the new one like. That's that's actually the right way to think about it. Isn't it because it's like you know we were trying to recognize. You know like which tomatoes were ripe and which ones weren't and like our new approach, you know the old approach, like fifty percent of the time.

More was like letting in the unripe Tomatoes, or you know, 50 percent. More of the time we were like accepting fraudulent customers, like that's a really big difference. So, just because, like this particular validation set, we can't really see six versus eight doesn't mean the 0.2 % different isn't important. It could be so. My kind of rule of thumb is that this, like this number of like how many observations you actually looking at. I want that generally to be somewhere higher than twenty-two. Why 22? Because 22 is the magic number where the t-distribution roughly turns into the normal distribution right. So, as you may have learned, the T distribution is, is the normal distribution for small data sets right, and so, in other words, once we have twenty-two of thing or more, it kind of starts to behave kind of normally, in both sense of the words like it's Kind of more stable and you can kind of understand it better. So that's my magic number when somebody says do I have enough of something I kind of start out by saying like do you have 22 observations of the thing of interest? So if you were looking at like Lyme cancer, you know - and you had a data set that had like a thousand people without lung cancer and 20 people with lung cancer. I'd be like, I very much doubt we're going to make much progress. You know, because we haven't even got 20 of the thing you want so ditto with a validation set.

If you don't have twenty of the thing you want, that is very unlikely to be useful or, if, like the at the level of accuracy, we need it's not plus or minus 20. It's just it's that. That's the point where I'm thinking like be a bit careful. So, just to be clear, you want 22 to be the number of samples in each set like in the validation, the test and the Train. So what I'm saying is like, if

there's, if there's less than 22 of a class in any of the sets, then it's it's going to get it's getting pretty unstable at that point right and so, like that's just like the first rule of thumb, but then what I would actually do is like start practicing what we learned about the binomial distribution or actually very weak distribution. So, what's the what is the mean of the binomial distribution of n samples and probability P n times, P? Okay, thank you and times P. Is that mean all right? So if you've got a 50 % chance of getting ahead and you toss it a hundred times on average you get 50 heads, okay and then what's the standard deviation and P 1 minus B? Okay, so these are like two numbers: the first number you don't have to remember. It's intuitively obvious. The second one is one that try to remember forevermore, because not only does it come up all the time, the people that you work with we'll all have forgotten it. So you'll be like the one person in the conversation who could immediately go. We don't have to run this 100 times. I can tell you straight away: it's binomial.

It's going to be NP q, NP 1 minus B. Then there's the standard error. The standard error is, if you run a bunch of trials each time getting a mean. What is the standard deviation of the mean? I don't think you guys are covered this yet. Is that right? No, so this is really important, because this means like, if you train a hundred models right each time. The validation set accuracy is like the meaning of a distribution, and so therefore, the standard deviation of that validation set accuracy. It can be calculated with the standard error, and this is equal to the standard deviation divided by square root. N all right. So this tells you so like one approach to figuring out, like is my validation set big enough is train your model five times with exactly the same hyper parameters each time and look at the validation set accuracy each time and give you know, there's like a mean And a standard deviation of five numbers you could use or a maximum and a minimum you can choose, but to save yourself some time you can figure out straight away that like okay! Well, I I have a point: nine nine accuracy as to you know whether I get the cat correct or not correct. So, therefore, the standard deviation is equal to 0.99 times 0.01. Okay and then I can get the standard error of that right. So so, basically the size of the validation set.

You need it's like, however big it has to be, such that your insights about accuracy, good enough for your particular business problem, and so, like I say, like the simple way to do. It is to pick a validation set of like a size of thousand trained five models and see how much the validation set accuracy varies and if it's like, if they're, if it's they're, all close enough for what you need, then you're fine, if it's not. Maybe you should make it bigger, or maybe you should consider using cross-validation instead, okay, so like, as you can see, it really depends on what it is you're trying to do, how common you're less common class is and how accurate your model is. Could you pass that back to Melissa, please? Thank you. I have a question about the less common classes. If you have less than 22, let's say you have one sample of something. Let's say it's a face and I only have one representation from that particular country. Do I toss that into the training set and it adds variety to I pull it out completely out of the data set, or do I put it in a test set instead of the validation set, so he certainly couldn't put it in the test of the validation Set because you're, asking kind of I mean in general because you're asking can I recognize something I've never seen before, but actually this this question of like can. I recognize something I've not seen before, there's actually a whole class of models specifically for that purpose, it's called either one-shot learning, which is you get to see something once and you have to recognize it again or zero shot learning, which is where you have to recognize Something you've never seen before we're not going to cover them in this course, but they can be useful for things like face recognition.

You know like is this the same person I've seen before and so generally speaking, obviously,

for something like that to work, it's not that you've never seen a face before it's that you've never seen Melissa's face before you know, and so you see Melissa's face once and You have to recognize it again yeah. So in general you know your validation, set and test set need to have the same mix or frequency observations that you're going to see in production in the real world. And then your training set should have an equal number in each class. And if you don't just replicate the less common one until it is equal, so this is, I think, we've mentioned this paper before a very recent paper that came out. They tried lots of different approaches to training with unbalanced datasets and found consistently that over sampling, the less common class until that is the same size as the more common class is always the right thing to do so. You could literally copy you know so, like I've. Only got a thousand, you know ten examples of people with cancer and 100. Without so I could just copy those 10 and other. You know 90 times that's kind of a little memory and efficient. So a lot of things, including I think, SK - learns random forests have a class weights parameter that says each time your boot strapping or resampling.

I want you to sample the less common class with a higher probability or did or if you do and doing deep learning you know make sure in your mini batch. It's not randomly sampled, but it's a stratified sample, so the less common class is picked more often. Okay, okay, so let's get back to finishing off our random forests, and so what we're going to do today is

3. 00:18:45

- **Back to Random Forest from scratch.**
- **“Basic data structures” reviewed**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

We're going to finish off writing our random forests and then, after day, you're after today, your homework will be to take this class and to add to it all of the random forest interpretation, algorithms that we've learned. Ok, so obviously to be able to do that. You're going to need to totally understand how this class works, so, please, you know, ask lots of questions as necessary as we go along. So just to remind you, we're doing the the bulldozers tackle competition data set again. We split it as before into 12,000 validation. The last 12,000 records and then just to make it easier for us to keep track of what we're doing we're going to just pick two columns out to start with year made and Machine hours on the meter, okay, and so what we did last time was. We started out by creating a tree ensemble and the tree ensemble had a bunch of trees, which was literally a list of entries trees where each time we just called create tree and create tree contained a sample size number of random indexes. Okay, this one is drawn without replacement, so remember, bootstrapping means sampling with replacement, so normally with scikit-learn. If you've got n rows, we grab n rows with replacement, which means many of them will appear more than once so each time we get a different sample, but it's always the same size as the original data set, and then we have our set our F samples, A function that we can use, which does with replacement sampling of less than n rows. This is doing something again, which is its sampling without replacement sample size, rows.

Okay, because we're permuting the numbers from naught to self dot y -1 and then grabbing the first self dot sample size of them, actually there's a faster way to do this. You can just use NPR and embrace, which is a slightly more direct way, but this way it works as well alright.

So this is our random sample for this one of our entries trees add so then we're going to create a decision tree and our decision tree. We don't pass it all of X, we pass it. These specific indexes and remember X, is a pandas data frame. So if we want to index into it with a bunch of integers, we to use iLok integer locations - and that makes it behave - indexing wise just like numpy now why vector is numpy, so we can just index into it directly and then we're going to keep track About minimum of each size, and so then the only other thing we really need an ensemble is somewhere to make a prediction, and so we were just going to do the mean of the tree prediction for each tree, all right. So that was that and so then, in order to be able to run that we need a decision tree class because it's being called here and so there we go okay. So that's the starting point. So the next thing we need to do is to flesh out our decision tree. So the important thing to remember is all of our randomness happened back here in the tree. Ensemble, the decision tree class, we're going to create, doesn't have randomness in it. Okay, so all right now we are building a random B regressor right.

So that's why we're taking the mean of the tree, the outputs? If we were to work with classification, do we take the max like the classifier will give you either zeros or ones? No, I would still take the mean so the so each tree is going to tell you what percentage of that leaf. Node contains cats and what percentage to take contains dogs. So then I would average all those percentages and say across the trees. On average, there is 19 % cats and 81 percent dogs good question, so you know random tree classifiers are almost identical or can be almost identical. The random tree regresses the technique. We're going to use to build this today will basically exactly work for a classification. It's certainly for binary classification. You can do with exactly the same code for multi-class classification. You just need to change your data structure, but so that, like you, have like a one, hot encoded matrix or a list of integers that you treat as a one hot encoded matrix okay. So our decision tree so remember our idea here is that we're going to like try to avoid thinking so we're going to basically write it as if everything we need already exists. Okay, so we know from when we created the decision tree, we're kind of pass in the X, the Y and the minimum leaf size. So here we need to make sure we've got the X and the y and the minimum left sides.

Okay, so then there's one other thing, which is, as we split our tree into sub trees, we're going to need to keep track of which of the row indexes, went into the left-hand side of the tree, which went into the right-hand side of the tree. Okay, so we're going to have this thing called indexes as well right, so at first we just didn't, bother, passing and indexes at all. So if indexes is not passed in, if it's none, then we're just going to set it to everything the entire length of Y right, so NP dot. A range is the same as just range in Python, but it returns an umpire rate right so that the root of a decision tree contains all the roads. That's the definition really of the root of a decision tree. So all the rows is Rho naught. Rho 1, Rho, 2, etc up to row y -1. Okay is going to store away all that information that we were given we're going to keep track of. How many rows are there and how many columns are there? Okay, so then the every leaf and every node in a tree has a value. It has a prediction that prediction is just equal to the average of the dependent variable okay, so every node in the tree Y indexed with the indexes is the values of the dependent variable that are in this branch of the tree. And so here is the main. Some nodes in a tree also have a score which is like how effective was the split here right, but that's only going to be true if it's not a leaf node right, a leaf.

Node has no further splits, and at this point, when we create a tree, we haven't done any splits yet so it's score starts out as being infinity. Okay, so having built that the root of the tree, our next job, is to find out, which variable should we split on and what level of that variable? Should we split on so let's pretend that there's something that does them find bass bit so then

we're done. Okay, so how do we find a variable to split on so well? We could just go through each potential. Variable so C contains the number of columns we have so go through each one and see if we can find a better split than we have so far on that column, okay, now notice. This is like not the full random forest definition. This is assuming that max features they're set to all right. Remember we could set max features too, like 0.5, in which case we wouldn't check all the numbers should not to see. We would check half the numbers at random from not to see so. If you want to turn this into like a random forest, that has the max features support, we could easily like add one line of code to do that, but we're not going to do it in our implementation today. So then we just need to find better split and since we're not interested in thinking at the moment for now we're just going to leave that empty alright, so there one other thing I like to do with my kind of word start: writing a class is I'd Like to have some way to print out, what's in that class, all right, and so if you type print followed by an object or if it Jupiter notebook, you just type the name of the object.

At the moment, it's just printing out underscore underscore main underscore underscore got decision tree at blah, blah blah, which is not very helpful right. So if we want to replace this with something helpful, we have to define the special Python method named dan direct crack to get a representation of this object. So when we type when we see please just write the name like this behind the scenes that calls that function and the default implementation of that method is just to print out this unhelpful stuff. So we can replace it by instead saying, let's create a format string where we're going to print out N and then show N and then print vowel and then show Val okay. So how many? How many rows are in this node and what's the average of the dependent variable? Okay, then, if it's not a leaf node, so if it has a split, then we should also be able to print out the score. The value we split out and the variable that we split on now, you'll notice here self dot is leaf, is leaf, is defined as a method, but I don't have any parentheses after it. This is a special kind of method, code of property, and so a property is something that kind of looks like a regular variable, but it's actually calculated on the fly. So when I call is leaf, it actually calls this function right, but I've got this special decorator property, okay, and what this says is basically, you don't have to include the parentheses when you call it okay, and so it's going to say all right. Is this a leaf or not so a leaf is something that we don't spit on.

If we haven't split on it, then it's score is still set to infinity. So that's my logic. That makes sense so this uh, this at notation, is called a decorator. It's basically a way of telling Python more information about your method. Does anybody here remember where you have seen decorators before we pass it over you yeah? Where have you seen that where have you seen decorators tell us more about flask and Wow yeah? What is that that no words so flasks, so anybody who's done any web programming before with something like flask or a similar framework would have had to have said, like this method is going to respond to this bit of the URL and either to post or to Get and you put it in a special decorator, so behind-the-scenes, that's telling Python to treat this method in a special way. So here's another decorator, okay, and so you know, if you get more advanced with Python, you can actually learn how to write your own decorators, which, as was mentioned, you know basically insert some additional code but for now just know, there's a bunch of predefined decorators. We can use to change how our methods behave, and one of them is a property which basically means you don't have to put parentheses anymore, which of course means you can't add any more parameters beyond self.y. If it's not belief, why is this for infinity? Because infinity mean you're at the root, why no infinity means that you're not at the root? It means you're at a leaf.

So the root will have a split assuming we find one, but everything will have a split till we get

all the way to the bottom and leaf, and so the leaves will have a score of infinity because they won't split great all right. So that's our decision tree, it doesn't do very much, but at least we can like create an ensemble right. Ten trees sample size, a thousand right and we can make print out. So now, when I go M trees, zero, it doesn't say blah blah blah blah blah. It says what we asked it to say: n called the thousand now 10.8. Oh wait, okay, and this is the leaf because we haven't spit on it. Yet so we've got nothing more to say, okay, so then the indexes are all the

4. 00:32:45

- **Single Branch**
- **Find the best split given variable with 'find_better_split', using Excel demo again**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Numbers from nought to a thousand okay, because the base of the tree has everything. This is like everything in the random sample that was passed to it because remember by the time we get to the point where it's a decision tree where we don't have to worry about any of the randomness in the random forest anymore. Okay, all right! So, let's try to write the thing which finds a split okay, so we need to implement, find better split, okay, and so it's going to take the index of a variable. Variable number one variable number three whatever, and it's going to figure out. What's the best blit point: is that better than any split we have so far and for the first variable the answer will always be yes, because the best one so far is none at all, which is infinity bad okay. So, let's start by making sure we've got something to compare to so the thing we're going to compare two will be scikit-learn, x', random forest, and so we need to make sure that psychic learns. Random forest gets exactly the same data that we have. So we start out by creating ensemble grab a tree out of it and then find out which particular random sample of x and y. Did this tree use okay and we're going to store them away so that we can pass them to scikit-learn. So we have exactly the same information, so let's go ahead and now create a random forest using scikit-learn. So one tree, one decision, no bootstrapping, so the whole the whole data set. That's oh.

This should be exactly the same as the thing that we're going to create this tree. Okay, so let's try so we need to define, find better split. Okay, so fine, better split takes a variable; okay, so let's define our x independent variables and say: okay. Well, it's everything inside our tree, but only those indexes that are in this node right, which at the top of the tree, is everything all right, and just this one variable okay and then for our Y's. It's just whatever a dependent variable is at the indexes in this node. Okay, so there's our X & amp Y. So let's now go through every single value in our independent variable and so I'll show you what's going to happen. So let's say our independent variable is um ade and not going to be an order right and so we're going to go to the very first row and we're going to say: okay, yeah mate here is three right, and so what I'm going to do is I'm Going to try and calculate the score if we decided to branch on the number three alright, so I need to know which rows are greater than three which rows are less than equal to three and they're, going to become my left-hand side, my right hand, side, but And then we need a score right, so there's lots of schools we could use so in random forests. We call this. The information gain right. The information gain is like how much better does our score get because we split it into these two groups of data. There's lots of ways we could calculate it.

Jinni cross-entropy root mean squared error whatever, if you think about it, there is an alternative formulation of root, mean squared error, which is mathematically the same to within a constant scale, but it's a little bit easier to deal with, which is we're gon na try and find A split which the causes the two groups to each have as lower standard, deviation as possible right so like. I want to find a spirit that puts all the cats over here and all the dogs over here right. So if these are all cats - and these are all dogs, then this has a standard deviation of zero, and this has a standard deviation of zero or else this is like a total around a mix of cats and dogs. This is a totally random mix of cats and dogs. They're going to have a much higher standard deviation, make sense, and so it turns out, if you find a split that minimizes those group standard deviations or specifically the weighted average of the true standard deviations. It's mathematically. The same as minimizing the root mean square error, that's something you can prove to yourself after class. If you want to okay, so we're going to need to find first of all split this into two groups, so where's all the stuff that is greater than three. So greater than three is this one, this one and this one? So we need the standard deviation of that.

So let's go ahead and say standard deviation of greater than three that one that one and that one okay and then the next will be the standard deviation of less than or equal to three. So that would be that one that one that one and then we just take the weighted average of those two and that's our score. That would be our score if we split on three that make sense, and so then the next step would be try to spit on four try spitting on one try spitting on six redundantly. Try splitting on four again redundantly, try spitting on one again and find out which one works best. So that's our code here is we're going to go through every row, and so let's say okay left hand. Side is any values in X that are less than or equal to this particular value. Our right hand, side is every value in X that are greater than this particular value. Okay, so what's the data type that's going to be in LHS and RHS? What are they actually going to contain they're going to be arrays arrays of what rays of erosive audience yeah, which we can treat a zero and one okay, so LHS will be an array of false every time, it's not less than or equal to and true otherwise, And RHS will be a boolean array of the opposite. Okay and now we can't take a standard deviation of an empty set right. So if there's nothing that's greater than this number, then these will all be false, which means the sum will be zero.

Okay - and in that case, let's not go any further with this step because there's nothing to take the standard deviation of, and it's obviously not a useful split. Okay. So assuming we've got this far, we can now calculate the standard deviation of the left-hand side and of the right-hand side and take the weighted average or the sums the same thing to us to a scaler right and so there's a score. And so we can then check is this better than our best score so far and our best score so far we initially initialized it to infinity right so initially this is. This is better. So if it's better, let's store away well, as the information we need, which variable has found this better split, what was the score we found and what was the value that we spit on? Okay, so there it is. So if we run that and I'm using time it so what time it does is that sees how long this command takes to run and it tries to give you a kind of statistically valid measure of that. So you can see here, it's run run at ten times to get an average and then it's done that seven times to get a mean and standard deviation across runs, and so it's taking me 75 milliseconds, plus or minus ten okay. So, let's check that this works find bladder split, tree zero, so zero is year made one is machine hours current meter, so I with one we got back machine hours, current meter, thirty, seven, four, four with this score and then we ran it again with zero. That's year made and we've got a better score: 658 and split 1974 and so 1974. Let's compare yeah.

That was what this treated as well. Okay, so we've got. We've confirmed that this method is doing is giving the same result that, as K loans, random forests did. Okay - and you can also see here the value 10 point - oh eight and again matching here the value. Ten point: oh eight, okay! So we've got something that can find once bit. Could you pass that to your net, please so, Jeremy? Why don't we put a unique on the eggs there because I'm not trying to optimize the performance yet, but do you see that no like he is doing more yeah, so it's like and you can see in the excel? I like checked this one twice. I check this four twice: unnecessarily: yeah, okay, so and so you're not already thinking about performance, which is good. So tell me: what is the computational complexity of this section of the code and and like ever think about it, but also like feel free to talk us through it? If you want to kind of think and talk at the same time, what's the computational complexity of this piece of code, can I pass it over there? Yes, all right, Jay take us through your thought process. I think you have to take each different values through the column, to calculate it once to see those splits so and then compare oh the cup, like all the possible combinations between these different values, so that can be expensive like this yours huh.

Can you do, somebody else would have tell us the actual computational complexity so like yeah, quite high Jayde's thinking how high I think it's great okay. So tell me: why is it N squared, Oh because for the full loop it is in yes? And I think I guess the standard deviation well ticket in so it's in square, okay or um. This one, maybe is even is yet to know like this is like which ones are less than X. I I'm gon na have to check every value to see if it's less than X, I okay, and so so it's useful to know like how do I quickly calculate computational complexity? I can guarantee most of the interviews you do are going to ask you to calculate computational complexity on the fly, and it's also like when you're coding. You want it to be second nature, so the technique is basically. Is there a loop, okay with then we're obviously doing this end times? Okay, so there's an N involved. It's there a loop inside the loop. If there is, then you need to multiply those two together in this case. There's not. Is there anything inside the loop? That's not a constant time thing, so you might see a sort in there and you just need to know that sort is n. Log n like that, should be second nature. If you see a matrix multiply, you need to know what that is in this case. There are some things that are doing element wise array operations right, so keep an eye out for anything where lump, I is doing something to every value of an array in this case is checking every value of x against a constant.

So it's going to have to do that n times so to flesh this out into a computational complexity. You just take the number of things in the loop and you multiply it by the highest computational complexity. Inside the loop n times, n is N squared and you pass them in this case. Couldn't we just pre sort the list and then do like 1 + log n computation? There's lots of things we can do to speed this up. So at this stage is just like what is the computational complexity we have, and but absolutely it's certainly not as good as it can be. Okay, so and that's where we're going to go next, just like alright N squared is not is not great. So, let's try and make it

5. 00:45:30

- **Speeding things up**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Better, so here's my attempt at making it better and the idea is this. Ok, who wants to first of all, tell me: what's the equation for standard deviation, Masha, can you grab the pose so for the standard deviation, it's the difference between the value and its mean. It's we take a square root of that, so that we take the the power of two. Then we sum up all of these situations and we take the square root out of all this sum. Yeah, you have to fight divided by M yep, yep great good. Okay, now, in practice we don't normally use that formulation because it kind of requires us calculating. You know, X, minus the mean lots of times. Does anybody know the formulation that just requires X and x? Squared anybody happen to know that one? Yes at the back? Can I pass that back there square root of a mean of squares, squared off mean yeah great mean of squares, minus the square of the means right. So that's a really good. 1/8. That's a really good one to know because, like you can now calculate variances or standard deviations of anything, you just have to first of all grab the column as it is the column squared right and as long as you've got those stored away somewhere, you can immediately Calculate the standard deviation, so the reason this is handy for us is that if we first of all sort our data right, let's go ahead and sort our data.

Then, if you think about it, as we kind of start going down one step at a time right, then each group, it's exactly the same as the previous group on the left hand, side with one more thing in it and on the right hand, side with one Less thing in it, so, given that we just have to keep track of some of X and some of x squared, we can just add one more thing to X, one more thing x, squared on the left and remove one thing on the right. Okay, so we don't have to go through the whole lot each time, and so we can turn this into a order n algorithm. So that's all I do here is I sort the data right and they're going to keep track of the count of things on the right, the sum of things on the right and the sum of squares on the right and initially everything's on the right hand, side. Okay, so initially n is the count. Y sum is the sum on the right and Y squared sum is the sum of squares on the right, and then nothing is initially on the left, so it's zeros, okay and then we just have to loop through each observation. Right and add one to the left hand, count subtract one from the left right hand, count add the value to the left hand, count subtract it from the right hand, count add the value squared, to the left hand, subtract it from the right hand. Okay. Now we do need to be careful, though, because if we're saying less than or equal to one say we're, not stopping here we're stopping here like we have to have everything in that group. So the other thing I'm going to do is I'm just going to make sure that the next value is not the same as this value.

If it is I'm going to skip over it right, so I'm just going to double check that this value and the next one aren't the same okay. So as long as they're, not the same, I can keep going ahead and calculate my standard deviation. Now passing in the count, the sum and the sum squared right and there's that formula: okay, the sum is squared divided by the square of the sum. So i minus the square of the sum. I do that's the right hand side, and so now we can calculate the weighted average score just like before, and all of these lames are now the same. Okay, so we've turned our order and square an algorithm into an order n algorithm and in general stuff like this, is going to get you a lot more value than like pushing something onto a spark cluster or ordering faster ram or using more cores and your cpu Or whatever right, this is the way you want to be. You know, improving your code and specifically write your code right without thinking too much about performance run. It is it fast enough for what you need then you're done, if not profile it right, so in Jupiter. Instead of seeing percent time at you say, % p run and it will tell you exactly where the time was spent in your algorithm and then you can go to the bit. That's actually taking the time and think about like okay is this: this is algorithmically as efficient as a can be okay.

So in this case, we run it and we've gone down from 76 milliseconds to less than 2 milliseconds, and now some people that are new to programming think like oh great, I've saved 60, something milliseconds. But the point is this: is going to get run like tens of millions of clients, okay, so the 76 millisecond version is so slow that it's got to be impractical for any random forest. You using in practice right where else the one millisecond version I found is actually quite quite acceptable and then check. The numbers should be exactly the same as before, and oh yeah, okay, so now that we have a function, find better split. That does what we want. I want to insert it into my decision tree class, and this is a really cool Python trick. Python does everything dynamically right, so we can actually say the method called find better split in decision tree. Is that function I just created and that might sticks it inside that class now I'll, tell you what's slightly confusing about this. Is that this thing this word here and this word here: they actually have no relationship to each other. They just happen to have the same letters in the same order right so like I could call this find better split, underscore foo right and then I could like call that right and call that right so now my function is actually called fine, better split, underscore foo, But my method, I'm expecting to call something called decision tree dot, fine, better split, all right, so here I could say decision tree dot, fine, better split equals, find better split, underscore foo! Okay, you see that's the same thing.

Okay, so like it's important to understand how namespaces work like in in every language that you use, one of the most important things is kind of understanding how how it figures out what a name refers to. So this here means find better split as to find inside this class right and nowhere else right. Well, I mean there's a parent class, but never mind about that. This one here means find better split. Fou in the global namespace, a lot of languages, don't have a global namespace that Python does okay, and so the two are like, even if they happen to have the same letters in the same order. They're not referring in any way to the same thing. That makes sense. It's like this family over here may. Have somebody called Jeremy and my family has somebody called Jeremy and our names happen to be the same, but we're not the same person, okay, great! So now that we've stuck the decision tree sorry, I did a fine Bettis flip method inside the decision tree with his new definition. When I now call the tree ensemble constructor all right. The decision tree ensemble instructor called create tree create tree instantiated decision tree decision tree called find vas whit, which went through every column to see if it could find a better split and we've now defined find better split and therefore tree ensemble when we create it has Gone ahead and done this wet that makes sense, don't have any anybody have any questions, uncertainties about that like we're, only creating one single split so far, all right, so this is pretty pretty neat right.

We kind of just do a little bit at a time testing everything as we go, and so it's as, as, as you all implement the random

6. 00:55:00

- **Full single tree**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Forest interpretation techniques you may want to try programming this way too, like every step check that you know what you're doing matches up with what scikit-learn does or with a test that you've built or whatever. So at this point, we should try to go deeper very inception. Right so let's go now max depth is two, and so here is what scikit-learn did after breaking it in

made 74, it then broke at Machine hours later 29:56, so we had this thing called find violet right, which just went through every column and try to see. If there's a better split there, all right, but actually we need to go a bit further than that. Not only do we have to go through every column and see if there's a better split in this node, but then we also have to see whether there's a better split in the left and the right sides that we just created right. In other words, the left right side and the right-hand side should become decision, trees themselves right. So there's no difference at all between what we do here to create this tree and what we do here to create this tree other than this one contains 159 samples, and this one contains a thousand. So this row of codes exactly the same as we had before. Right and then we check it actually, we could do this a little bit easier. We could say if self dot is leaf. Right would be the same thing. Hey don't leave it here for now, so it's self dot score. So if the score is infinite.

Still, let's write it properly, yes, wait! So: let's go back up and just remind ourselves is leaf. Is self that's poor equals in okay. So since it's there, we mostly use it. So if it's a leaf node, then we have nothing further to do right. So that means we're right at the bottom. There's no split. That's been made okay, so we don't have to do anything further. On the other hand, if it's not a leaf node, so it's somewhere back earlier on, then we need to split it into the left hand. Side and the right hand side now earlier on, we created a left hand side in the right hand, side, array of bullying's right now, better would be to have here. We have an array of indexes and that's because we don't want to have a full array of all the volumes in every single node right because remember, although it doesn't look like there are many nodes, when you see a tree of this size when it's fully expanded, The bottom level, if there's a minimum leaf size of one, contains the same number of nodes as the entire data set, and so, if every one of those contained a full boolean array of size of the whole data set, you've got squared memory requirements which would be Bad right, on the other hand, if we just store the indexes there for things in this node and that's going to get smaller and smaller, okay, so NP non.

Zero is exactly the same as just this thing, which gets the boolean array, but it turns it into the indexes of the truths okay, so this is now a list of indexes for the left-hand side and indexes to the right-hand side. Alright. So now that we have the indexes the left-hand side and the right-hand side, we can now just go ahead and create a decision tree. Okay, so there's a decision tree for the left and there's our decision tree for the right. Okay - and we don't have to do anything else - we've already written these - we already have a function of a constructor that can create a decision tree so like when you really think about what this is doing. It kind of hurts your head right, because the reason the whole reason that fine vast bit got called is because find vasp lit is called by the decision tree constructor. But then the decision tree that then find vast bit itself then causes the decision tree constructor. So we actually have circular recursion and I'm not nearly smart enough to be able to think through recursion. So I just choose not to write like I just write what I mean and then I don't think about it anymore. Right like what do I want well to find a variable-speed I've got to go through a few column, see if there's something better. It had managed to do a split figure out left-hand side of the right-hand side and make them into decision trees, okay, but now try to think through how these two methods call each other would just drive me crazy, but I don't need to write.

I know I have a decision tree constructor that works no. No, no. I have a vine up find basket that works. So that's it right. That's how I do recursive programming is by pretending. I don't I just just ignore it. That's my advice. A lot of you are probably smart enough to be able to think through it better than I can. So that's fine! If you can all right so now that I've written that again, I can patch it into the decision tree class and as soon as I do, the tree ensemble constructor will now use that right because pythons dynamic right, that's just happens

automatically. So now I can check my left-hand side should have 159 samples right and a value of nine point. Six six there. It is 159 samples, nine point, six six right hand: side, huh, 841, 10.15, the left hand, side of the left hand, side, 150, samples, nine point, six to 150 samples; nine point: six! Okay, so you can see like I'm, because I'm not nearly clever enough to write machine learning algorithms like not only can I not write them correctly. The first time, often like every single line, I write, will be wrong right. So I always start from the assumption that the the line of code I just typed is almost certainly wrong and I just have to see why and how right and so like. I just make sure, and so eventually I get to the point where, like much to my surprise, it's not broken

7. 01:01:30

- **Predictions with 'predict(self,x)',**
- **and 'predict_row(self, xi)'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Anymore, you know so here I can feel like okay, this. It would be surprising if all of these things accidentally happen to be exactly the same as scikit-learn, so this is looking pretty good. Okay, so now that we have something that can build a whole tree where you want to have something that can calculate predictions right and so remind you, we already have something that calculates predictions for a tree ensemble by calling trade-up predict. But there is nothing called treetop predict, so we're gon na have to write that to make this more interesting. Let's start bringing up the number of columns that we use. Let's create our tree ensemble again and this time let's go to a maximum depth of three okay. So now our tree is getting more interesting and, let's now define how do we create a set of predictions for a tree, and so a set of predictions for a tree is simply the prediction for a row for every row. That's it all right, that's our predictions! So the predictions for a tree are every rows predictions in an array. Okay, so again we're like skipping thinking, thinking's hard, you know so, let's just like keep pushing it back. This is kind of Handy right notice that you can do four in array with an umpire array, regardless of the rank of the array, regardless of the number of axes in the array and what it does is. It will look through the leading axis.

At least these concepts are going to be very, very important as we get into more and more neural networks, because we're going to be all doing, tensor computations all the time. So the leading axis that the vector is the vector itself, the leading axis of a matrix, are the rows. The leading access axis of a three dimensional tensor, the matrices that represent the slices and so forth right. So, in this case, because X is a matrix, this is going to look through the rows and if you write your kind of tensor code, this way then it'll kind of tend to generalize nicely to higher dimensions like it doesn't really mention matter. How many dimensions are in X? This is going to loop through each of the leading axis okay, so we can now call that decision tree. Do I predict alright? So all I need to do is write, predict row right and I've delayed thinking so much, which is great that the actual point where I actually have to do the work - it's now basically trivial. So, if we're at a leaf, no, then the prediction is just equal to whatever that value was which we calculated right back in the original tree. Constructor is to assist the average of the Y's right. If it's not a leaf node, then we have to figure out whether to go down the left-hand path or the right-hand path to get the prediction right. So if this variable in this row is less than or equal to that thing, we decided the amount we decided to split on. Then we go down the left path.

Otherwise we go down the right path: okay and then having figured out what path we want which tree we want, then we can just call predict row on that. Right and again, we've accidentally created something recursive again. I don't want to think about how that works. Control flow wise or whatever, but I don't need to because, like I just it just does like, I just told it what I wanted so I'll trust it to work right if it's a leaf return, the value otherwise return. The prediction for the left, hand, side or the right hand side as appropriate. There notice this here this, if has nothing, to do with this, if all right this, if is a control flow statement that tells Python to go down on that path or that path to do some calculation, this, if is an operator that returns a value, so those Of you that have done C or C++ will recognize it as being identical to that. It's called the ternary operator. All right, if you haven't that's fine, basically, what we're doing is we're going to get a value where we're going to say it's this value. If this thing is true and that value otherwise - and so you could write it this way right, but that would require writing four lines of code to do one thing and also require you to code that, if you read it to yourself or to somebody else, is Not at all, naturally, the way you would express it right, I want to say the tree I going to go down is the left-hand side, if the variables less than the split or the right-hand side, otherwise right, so I want to write my code the way I Would think about all the way I would say my code, okay, so this kind of ternary operator can be quite helpful for that.

Alright. So now that I've got a prediction for a row, I can dump that into my class, and now I can create calculate predictions, and I can now plot my actuals against my predictions. When you do a scatter plot, you'll often have a lot of dots sitting. On top of each other, so a good trick is to use alpha. Alpha means how transparent the things not just a map plot lib but like in every graphics package in the world pretty much, and so, if you set alpha to less than 1, then this is saying you would need 20 dots on top of each other for it To be fully blue, and so this is a good way to kind of see how much things are sitting on top of each other. So it's a good trick but trick the scatter plots. There's my R squared, not bad, and so let's now go ahead and do a random forest with no max mana spitting and our tree ensemble had no max amount of spitting. We can compare our R squared there are squared and so they're not the same, but actually ours is a little better. So I don't know what we did differently, but we'll take it okay, so we have now something which, for a forest with a single tree in is giving as good accuracy on a validation set using an actual real-world data set. You know books for pluto's is compared to scikit-learn, so let's go ahead and round this out. So what I would want to do now is to create a package that has this coding, and I created it by like creating a method here - a method here, a method here and catching them together.

So what I did with now is: I went back through in

8. 01:09:05

- **Putting it all together,**
- **Cython an optimising static compiler for Python and C**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

My notebook and collected up all the cells had implemented methods and pasted them all together right and I've just pasted them down here. So here's this is my original tree ensemble and here is all the cells and the decision tree. I just dumped them all into one place without any change, so that was it. That was the code we wrote together so now I can go ahead and I

can create a tree ensemble. I can calculate my predictions. I can do my scatter plot. I can get my r-squared right and this is now with five trees right and here we are. We have a model of blue dog for bulldozers, with a 71 %, a squid with a random forest. We wrote entirely from scratch. That's pretty cool any questions about that and I know there's like quite a lot to get through. So I during the week feel free to ask on the forum about any bits of code you come across. Can somebody pass the box to Mercia others? Can we get back to the probably to the top, maybe a decision tree when we said the score equal to infinity right? Yes, I do a calculator Scott, the score for the I mean, like I lost track of that, and specifically, I wonder when we implement when we implement, find VAR split, we check for self score equal to whether it's equal to infinity or not. It says to me: it seems like I'm clear whether we fall out of this. I mean like if we ever implement the method, if, if our initial value is infinity, so okay, let's talk sure the logic.

So so the decision tree starts out with a score at infinity, so in other words, at this point, when we've created the mode, there is no split, so it's infinitely bad. Okay, that's why the score is infinity and then we try to find a variable and a split that is better and do that we look through each column and say: hey column. Do you have a split which is better than the best one we have so far, and so then we implement that let's do the slow way since it's a bit simpler, find better split. We do that by looping through each row and finding out this is the current score. If we split here, is it better than the current score? The current score is infinitely bad. So, yes, it is, and so now we set the new score equal to what we just calculated and we keep track, of which variable we chose and the split we spit on. Okay, no worries; okay, great, let's take a five-minute break and I'll see you back here at 22. So when I tried comparing the performance of this against scikit-learn, this is quite a lot slower and the reason. Why is that, although, like a lot of the works being done by numpy, which is nicely optimized, c-code think about like the very bottom level of a tree? If we've got a million data points and the bottom level of the tree has something like 500 thousand decision points with a million leaves underneath right, and so that's like five hundred thousand split methods being called each one of contained, which contains multiple calls to numpy, which Only have like one item that's actually being calculated on, and so it's like that's like very inefficient and it's the kind of thing that Python is particularly not good at performance.

Wise right, like calling lots of functions, lots of times I mean we can see it's. It's not bad right, you know, for a kind of a random forest which 15 years ago would have been considered pretty big. This would be considered pretty good performance right, but nowadays this is some hundreds of times at least slower than than it should be. So what the scikit-learn folks did to avoid this problem was that they wrote their implementation in something called siphon and siphon is a superset of Python. So any Python you've written pretty much. You can use as siphon right, but then what happens is siphon runs it in a very different way, rather than passing it to the kind of the Python interpreter it instead converts it to C, can Kyle's that and then runs that C code right, which means the First time you run it, it takes a little long work, so it has to go through the kind of translation and compilation, but then, after that it can be quite a bit faster, and so I want to just to quickly show you what that looks like, because You are absolutely going to be in a position where siphons going to help you with your work and most of the people you're working with will have never used. It may not even know it exists, and so this is like a great superpower to have so to use siphon in a notebook. You say, load next load, extension siphon right and so here's a Python function bit. One here is the same as a siphon function is exactly the same thing with percent % at the top.

This actually runs about twice as fast as this right, just because it does their compilation. Here

is the same version again where I've used a special siphon extension called C death, which defines the C data type of the return value and of each variable right, and so, basically, that's the trick that you can use to start making things run quickly right and At that point now it knows it's not just some Python object called T. In fact I probably should put one here as well. Let's try that so we've got fib 2. We call that 53, so 453 yeah. So it's exactly the same as before, but we say what the data type of the thing we passed to it was is and then define the data types of each of the variables. And so then, if we call that okay, we've now got something that's 10 times faster right, so yeah, it doesn't really take that much extra and it's just it's just Python with a few little bits of markup. So that's like it's it's good to know that that exists, because if there's something custom you're trying to do, it's actually a find it kind of painful having to go out and you know going to see and compile it and whip it back and all that. Where else doing it here is pretty easy, can you pass that just your right, please not sure so when you're doing like for the second version of it, so in the case, an array for an NP array, this is a specific C type of yeah.

So there's like a lot of um specific stuff for integrating scythe on with numpy and there's a whole page about it yeah. So we won't worry about going over it, but you can read that and you can basically see the basic ideas. There's this C import which basically imports a certain types of Python library into the kind of the C bit of the code, and you can then use it in your siphon yeah. It's it's pretty straightforward! Well, good!

9. 01:18:01

- **“Your mission, for next class, is to implement”:**
- **Confidence based on tree variance,**
- **Feature importance,**
- **Partial dependence,**
- **Tree interpreter.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Question, thank you all right, so your your mission now is to implement confidence based on tree variance, feature importance, partial dependence in tree interpreter for that random, first, removing redundant features doesn't use a random forest at all. So you don't have to worry about that. The extrapolation is not an interpretation technique, so you don't have to worry about that, so it's just the other ones. So confidence based on tree variance, we've already written that code. So I suspect that the exact same code we would have in the notebook should continue to work, so you can try and make sure it get that working feature. Importance is with the variable shuffling technique, and once you have that working partial dependence, it will just be a couple of lines of code away rather than you know, rather than shuffling a column you're, just replacing it with a constant value that it's nearly the same code And then tree interpreter it's going to require you writing some code and thinking about that well, ince you've, written tree interpreter you're, very close if you want to to creating the second approach to feature importance, the one where you add up the importance across all of the Rows, which means you would then be very close to doing interaction importance, so it turns out that that there are actually there's actually a very good library for interaction importance for extra boost, but there doesn't seem to be one for random forests, so you could, like start By getting it working on our version and if you want to do interaction importance and then you could like get it working on the original site, scikit-learn version, and that would

be a cool contribution. All right, like sometimes writing it against your own implementation, is kind of nicer because you can see exactly what's going on all right.

So that's that's your job! Now you don't have to rewrite the random forest feel free to. If you want to you, know practice. So if you get stuck at any

10. 01:20:15

- **Reminder: How to ask for Help on Fastai forums**
- **http://wiki.fast.ai/index.php/How_to_ask_for_Help**
- **Getting a screenshot, resizing it.**
- **For lines of code, create a "Gist", using the extension 'Gist-it' for "Create/Edit Gist of Notebook" with 'nbextensions_configurator' on Jupyter Notebook, 'Collapsible Headings', 'Chrome Clipboard', 'Hide Header'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Point you know ask on the forum right: there is a whole page here on wiki dot fast play I about how to ask for help. So when you ask your co-workers on slack for help, when you ask people in a technical community on github or discourse for help or whatever asking for help, the right way will go a long way towards you know having people want to help you and be able To help here right so so, like search for your aunt's, like search for the arrow you're, getting see if somebody's already asked about it, you know how have you tried to fix it already? What do you think's going wrong? What kind of computer are you on? How is it set up? What are the software versions exactly? What did you type at exactly what happened right now? You could do that by taking a screenshot, so you know make sure you've got some screenshot software, that's really easy to use. So if I were to take a screenshot, I just hit a button. Select the area copy to clipboard, go to my forum, paste it in and there we go right. That looks a little bit too big. So, let's make it a little smaller all right, and so now I've got a screenshot. People can see exactly what happened better still, if there's a few lines of code and error messages to look at and create a gist. It just is a handy little github thing which basically lets you share code. So if I wanted to create a gist of this, I actually have a extension area that little extension.

So if I click on here, give it a name say: make public okay, and that takes my Jupiter notebook shares it publicly. I can then grab that URL copy link, location right and paste it into my forum post right and then, when people click on it, then they'll immediately see my notebook when it renders okay. So that's a really good way now that particular button is an extension. So on Jupiter, you need to click, end the extensions and click on just it right while you're there. You should also click on collapsible headiness. That's this really handy thing. I use that lets me collapse things and open them up. If you go to your Jupiter and don't see this MB extensions button, then just Google for Jupiter and B extensions it'll show you how to pip, install it and and get it set up right where those two extensions are

11. 01:23:15

- **We're done with Random Forests, now we move on to Neural Networks.**
- **Random Forests can't extrapolate, it just averages data that it has already seen, Linear Regression can but only in very limited ways.**

- **Neural Networks give us the best of both worlds.**
- **Intro to SGD for MNIST, unstructured data.**
- **Quick comparison with Fastai/Jeremy's Deep Learning Course.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Superduper handy alright, so other than that assignment, where we're done with random forests and until the next course, when you look at gPMs, we're done with decision tree ensembles and so we're going to move on to neural networks broadly defined, and so Dero networks are going to Allow us to to go beyond just you know: the kind of nearest neighbors approach of random forests. You know all around and forests can do - is to average data that it's already seen it can't extrapolate it can't. They can't calculate right, linear regression can calculate and can extrapolate, but only in very limited ways. Neural nets give us the best of both worlds, we're going to start by applying them to unstructured data all right, so unstructured data means like pixels or the amplitudes of sound waves or words. You know data where everything in all the columns are all the same type. You know, as opposed to like a database table where you've got like a revenue and a cost and a zip code and a state. It should be structured data, we're going to use it for structured data as well, but we're going to do that a little bit later. So structured data is a little easier and it's also the area which more people have been applying deep learning to for longer the. If you're doing the deep learning course as well, you know you'll see that we're going to be approaching kind of the same conclusion from two different directions.

So the deep learning course is starting out with big, complicated convolutional neural networks being solved with you know, sophisticated optimization schemes and we're going to kind of gradually drill down into like exactly how they work. Where else, with the machine learning course we're going to be starting out more with like how does stochastic gradient descent actually work, what do we do? What can we do is like one single layer which would allow us to create things like logistic regression when we add regularization to that? How does that give us things like Ridge, regression, elastic net lasso and then, as we add additional layers to that? How does that? Let us handle more complex problems, and so we're not going to we're only going to be looking at fully connected layers in this machine learning course, and then I think next semester, with your net you're, probably going to be looking at some more sophisticated approaches and so Yeah, so this machine learning we're going to be looking much more at like what's actually happening with the matrices and how they actually calculated and the deep learning it's much more like what are the best practices for actually solving. You know at a world-class level, real-world deep learning problems right so next week we're going to be looking at like the classic Emnes problem, which is like how do we recognize digits now, if you're interested you can like skip ahead and like try and do this with A random forest and you'll find it's not bad, but it given that a random forest is basically a type of nearest neighbors right. It's finding like what are the nearest neighbors in entry space, then a random forest could absolutely recognize that this nine, those pixels you know are similar to pixels we've seen in these other ones and on average they were nines as well right so like it can absolutely Solve these kinds of problems to an extent using random forests, but we end up being rather data limited, because every time we put in another decision point, you know we're having our data roughly, and so this is this limitation and the amount of calculation that we can Do where else with neural nets, we're going to be able to use lots and lots and lots of parameters using these tricks? We don't learn about with regularization, and so we're going to be able to do lots of computation and there's got to be very little limitation on

really what we can actually end up calculating as a result.

Great good luck with your random forest interpretation, and I will see you next time.

Lesson 08

Outline

- Building a Decision Tree from scratch
- Optimizing and comparing to SKlearn
- How to do 2 levels of decision trees
- Fleshing out the RF predict function
- Assembling our own decision tree
- Cython

Video Timelines and Transcript

1. 00:00:45

- **Moving from Decision Trees Ensemble to Neural Nets with Mnist**
- **lesson4-mnist_sgd.ipynb notebook**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So I I don't want to embarrass Rachel, but I'm very excited that Rachel's here. So this is Rachel for those of you that don't know she's not quite back on her feet after her illness, but well enough to at least come to at least part. With this lesson so worried if she can't stay for the whole thing and I'm really glad she's here, because Rachel actually wrote the vast majority of the lesson we're gon na see it's, I think it's a really really cool work. So I'm glad she's gon na at least see it being taught, even if unfortunately, she's not teaching it herself so good Thanksgiving present best for Thanksgiving present. So where, as we discussed at the end of last lesson, we're kind of moving from the C decision tree ensembles to two neural Nets, broadly defined and as we discussed you know, random forests and decision trees are limited by the fact. In the end that they're, basically they're basically doing nearest neighbors right, you know that all they can do is to get return the average of a bunch of other points, and so they can't extrapolate out to you know if you're thinking what happens if I'll increase my Prices by 20 % and you've never priced at that level before or what's going to happen to sales next year, and obviously, we've never seen next year before it's very hard to extrapolate. It's also hard, if it needs to you, know like it, can only do around log base two n decisions, you know, and so, if there's like a time series, it needs to fit too.

That takes like four steps to kind of get to the right time area. Then suddenly, there's not many decisions left for it to make. So it's kind of this limited amount of computation that it can do so. There's a limited complexity of relationship that it can model yes Prince. Can I ask about one more drawback of random forests yeah. So if we have a data as categorical variable, which are not in sequential order so for random forests, we and coat them and treat them as numbers. Let's say we have 20 cardinality and 1 to 20. So the result at random for our skills is like the spirit and important skills is something like less than 5 less than 6. But if the

categories are not sequential not in any order, what does that mean yeah? So so, if you've got like, oh, let's go back to bulldozers, erupts erupts with AC erupts na, I don't know whatever right and we arbitrarily label them like this right, and so actually we know that all that really mattered was if it had air conditioning. So, what's going to happen? Well, it's basically going to say like okay, if I group it into those together and those together that, like that's an interesting break just because it so happens that the air conditioning ones are going to end up in the right hand, side and then having done that. Right, it's then going to say: okay well within the group, with the two and three, it's going to notice that it's furthermore going to have to split it into two more groups.

So eventually it's going to get there it's going to pull out that category. It's just! It's going to take more splits than we would ideally like, so it's kind of similar to the fact that they get to model a line. It can only do it with lots of splits and only approximately let's, just fine, with categories that are not sequential, also yeah. So I can't do it. It's just like in some way it's sub-optimal because we just need to do more breakpoints than we would have liked, but it gets there. It does a pretty good job, and so, even although random forests, you know, do you have some deficiencies? They're, incredibly powerful, you know, particularly because they have so few assumptions they really had to screw up, and you know it's kind of hard to actually win Kaggle competition with a random first, but it's very easy to get like 10 %. So in like in real life, where often that third decimal place doesn't matter, random, forests are often like what you end up doing. But for some things like this Ecuadorian groceries competition, it's very very hard to get a good result with a random forest because, like there's a huge time series component and like nearly everything, is these two massively high cardinality categorical variables, which is the store and the item And like so this, so there's very little there to even throw at a random forest and the you know, the difference between every pair of stores is kind of different in different ways. And so you know there are some things that are just hard to get.

Even relatively good results for the random forest, another example is recognizing numbers you can get like okay results with a random forest, but in the end, they're kind of the relationship between you know like that. The spatial structure turns out to be important right and you kind of want to be able to do like computations like finding edges or whatever that kind of carry forward through through the computation. So you know just doing a kind of a clever nearest neighbors like a random forest. You know turns out not to be ideal. So if it's stuff, like this neural networks, turn out that they are ideal, neural networks turn out to be something that works particularly well. For both things, like the Ecuadorian, groceries, competition, so forecasting, sales over time, buy, store and buy item and for things like recognizing digits and for things like turning voice into speech, and so it's kind of nice between these two things: neural nets and random forests. We kind of cover the territory right. I don't. I haven't needed to use anything other than these two things for a very long time and will actually learn. I don't know them what course exactly, but at some point we'll learn also how to combine the two, because you can combine the two in really cool ways. So here's a picture from Adam guide key of an image. So an image is just a bunch of numbers. Right and each of those numbers is not to 255, and the dark ones are too close to 255 white ones are close to zero.

All right, so here is an example of a digit from this amnesty data set amnesties are really old. It's like a hello world of machine of neuro-networks, and so here's an example, and so there are 28 by 28 pixels if it was color. There would be three of these one for red one for green okay, so our job is to look at. You know the array of numbers and figure out that this is the

number eight which is tricky right. How do we do that, so we're going to use a few, a small number of fastai pieces and we're gradually going to remove more and more and more until by the end, we'll have implemented our own euro network from stretch our own training loop from scratch? And our own matrix multiplication from scratch, so we're gradually going to dig in further and further alright. So the data for amnesty, which is the name of this very famous data, set, is available from here, and we have a thing in fast: AI dot, IO called get data which will grab it from a URL and store it from your on your computer. Unless it's already there, in which case it'll, just go ahead and use it okay and then we've got a little function here, called load em nest which simply loads

2. 00:08:20

- **About Python 'pickle()' pros & cons for Pandas, vs 'feather()',**
- **Flatten a tensor**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

It up you'll see that it's zipped, so we could just use pythons gzip to open it up, and then it's also pickled. So if you have any kind of Python object at all, you can use this built-in etham library called pickle to dump it out onto your disk, share it around load it up later and you get back the same Python object. You started with so you've already seen this something like this with, like pandas, feather format. Right pickle is not just for pandas, it's not just for anything. It works for basically nearly every object so which might lead to the question. Well, why didn't we use pickle for a pandas data frame right and the answer is pickle works for nearly every Python object, but it's probably not like optimal for nearly any Python object right. So because, like we were looking at pandas dataframes with like over a hundred million rows, we really want to save that quickly and so feather is a format. That's specifically designed for that purpose, and so it's going to do that really fast. If we tried to pickle it, it would have been taken a lot longer. Right also note that pickle files are only for Python, so you can't give them to somebody else. Where else like a feather file, you can hand around yeah. So it's worth knowing that pickle exists, because if you've got some dictionary or some kind of object floating around that you want to save for later or send to somebody else, you can always just pickle it.

Okay. So in this particular case, the folks at deep learning network kind enough to provide a pickled version. Pickle has changed slightly over time and so old pickle files like this one you actually have to this - was a Python to one. So you have to tell it that it was encoded using this particular Python, 2 character set, but other than that Python, 2 & amp 3. You can normally open each other's pickle files all right so once we floated that in we loaded in like so, and so this thing, which we're doing here this is called D structuring and so D structuring means that load M nest is giving us back a couple Of tuples, and so if we have, on the left hand, side of the equal sign a couple of tuples, we can fill all these things in so we're given back a couple of training data, a couple of validation data and a couple of test data. In this case, I don't care about the test data, so I just put it into a variable called score, which kind of by like people in pick Python people tend to think of underscore as being a special variable, which we put things we're going to throw away Into it's actually not special, but it's just it's really common. If you see something assigned to underscore, it probably means you're, just throwing it away right by the way in a jupiter notebook, it does have a special meaning, which is the last cell that you calculate is always available in underscore by the way. But that's kind

of a separate issue.

So then, the first thing in that topple is itself at Apple and so we're going to stick that into X & amp Y for our training data and then the second one goes into X & amp Y for our validation data. Okay, so that's called destructuring and it's pretty common in lots of languages. Some languages don't support it, but those that do life becomes a lot easier. So as soon as I you know, look at some new data set, I just check out. What's what if I got all right so what's its tight? Okay, it's an umpire right, what's its shape, it's 50,000 by seven, eight four and then what about the dependent variables? That's an array. Its shape is 50,000, so this image is not of length. Seven, eight! Four! It's at size, 28 by 28. So what happened here? Well, we could guess, and we can check on the website. It turns that we would be right that all they did was. They took the second row and concatenate it to the first row and the third row and concatenate it to that and the fourth row and competitive's of that. So, in other words, they took this whole 28 by 28 and flattened it out into a single 1d array. That makes sense so it's going to be of size 28 squared. This is not like normal by any means. So don't think, like everything you see is going to be like this most of the time when people share images they share them as JPEGs or PNG s, you load them up.

You get back a nice 2d array, but in this particular case, for whatever reason, the thing that they pickled was flattened out to be 784, and this word flatten is very common with you know: kind of working with tenses. So when you flatten a tensor, it just means that you're turning it into a a lower rank tensor than you start up with. In this case, we started with a rank two tensor and the matrix for each image, and we turned each one into a rank. One tensor ie a vector so overall, the whole thing you know is a rank two matrix for

3. 00:13:45

- **Reminder on the jargon: a vector in math is a 1d array in CS,**
- **a rank 1 tensor in deep learning.**
- **A matrix is a 2d array or a rank 2 tensor, rows are axis 0 and columns are axis 1**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

A rank two tensor, rather than a rank three tensor. So just to remind us of you know the jargon here this and math. We would call a vector right in computer science. We would call it a 1d array, but because deep learning have people have to come across as smarter than everybody else, we have to call this a rank: 1 tensor, okay, they all mean the same thing more or less, unless you're, a physicist, in which case this Means something else and you get very angry at the deep learning people, because you say it's not a tensor. So there you go, don't blame me. This is just what people say. So this is either a matrix or a 2d array or a rank two tensor, and so once we start to get into three dimensions, we start to run out of mathematical names right, which is why we start to be nice just to say, rank three tensor and So there's actually nothing special about vectors and matrices that make them in any way more important than rank three tenses or rank for tensors or whatever. So I try not to use the terms vector and matrix where possible, because I don't really think they're there any more special than any other rank of tensor. Okay, so kind of it's good to get used to thinking of this as a rank, two tensor, okay and then the the rows and columns. If it was a if we're computer science people, we would call this dimension zero and dimension one.

But if we're deep learning people, we would call this axis zero or access one, okay and then just to be really confusing if you're an image person. This is the first axis, and this is the second axis all right. So if you think about like TVs, you know 1920 by 1080, columns by rows everybody else, including deep-learning and mathematicians rows by columns. So this is pretty confusing. If you use like the Python imaging library you get back columns by rows, pretty much everything else rows by columns, so be careful because they hate us because they're bad people. I guess I mean there's a lot of just um, particularly in deep learning like a whole. Lot of different areas have come together like information theory, computer vision, statistics signal, processing and you've ended up with this hodgepodge of nomenclature in deep learning, often like every version of things will be used. So today we're going to hear about something. That's called either negative log likelihood or binomial or categorical cross entropy, depending on where you come from, we've already seen something that's called either one hot encoding or dummy variables, depending on where you come from. It really is just like the same concept gets kind of somewhat independently invented in different fields and eventually they find their way to machine learning, and then we don't know what to call them. So we call them all of the above. Something like that.

So I think that's what's happened with with computer vision, rows and columns. So there's this idea of normalizing data, which is subtracting out the mean and dividing by the standard deviation. So a

4. 00:17:45

- **Normalizing the data: subtracting off the mean and dividing by stddev**
- **Important: use the mean and stddev of Training data for the Validation data as well.**
- **Use the 'np.reshape()' function**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Question for you do like often it's important to normalize the data so that we can more easily train a model. Do you think it would be important to normalize the independent variables for a random forest if we're training a random forest, be honest? I don't know why. We don't need to normalize. I just know that we've done okay. Does anybody want to think about? Why kara it wouldn't matter, because each scaling and transformation we can have will be applied to each row and we will be computing means, as we were doing like local averages and at the end we will of course want to denormalize it back to give. So it wouldn't change the results I'm doing about the independent variables, not the dependent variable. I taught to us. The world depend: okay, let's have a go Matthew, it might be because we just care about the relationship between the independent variables, an independent variable, so scale doesn't matter. Okay come on cat. Why? Why? Well? Why do we own? We could like, because at each split point we can just divide to see which, regardless of what scale you're on what minimizes variance, and that would right so really. The key is that, when we're deciding where to split all that matters is the order like it all. That matters is how they're sorted. So if we divide by the subtract the mean and divide by the standard deviation, they're still sorted in the same order. So remember when we implemented the random first, we said sort them and then we liked it.

Then we completely ignored the values we just said, like now add on one thing from the

dependent at a time, so so random forests only care about the sort order of the independent variables they don't care at all about their size. And so that's why they're wonderfully immune to outliers, because they totally ignore the fact that it's an outlier they only care about which one's higher than what other thing right. So this is an important concept. It doesn't just appear in random forests. It occurs in some metrics as well, for example, area under the ROC curve. You come across a lot that area under the ROC curve completely ignores scale and only cares about sort. We saw something else when we did the dendrogram Spearman's correlation is a rank. Correlation only cares about order not about scale so random forests. One of the many wonderful things about them are that we can completely ignore a lot of these statistical distribution issues, but we can't for deep learning because for deep learning, we're trying to train a parameterised model. So we do need to normalize our data. If we don't, then it's going to be much harder to create a network that trains effectively. So we grab the mean and the standard deviation of our training data and subtract out the mean divided by the standard deviation, and that gives us a mean of 0 and a standard deviation of 1. Now, for our validation data, we need to use the standard deviation and mean from the training data right.

We have to normalize it the same way. Just like categorical variables. We had to make sure they had the same indexes, mapped to the same levels for a random forest or missing values. We had to make sure we have the same median used when we were replacing the missing values. You need to make sure anything you do in the training set. You do exactly the same thing and the test and validation set. So here I'm subtracting out the training set me in the training set standard deviation. So this is not exactly zero. This is not exactly one, but it's pretty close, and so in general, if you find you try something on a validation set or a test set, and it's like much much much worse than your training set. That's probably because you normalized in an inconsistent way or encoded category is an inconsistent way or something like that all right. So let's take a look at some of this data, so we've got 10,000 images in the validation set and each one is a rank. One tensor of length: seven, eight, four. In order to display it, I want to turn it into a rank: two tensor of 28 by 28, so there's a dump, a has a reshape function that takes a tensor in and reshapes it to whatever size. Tensor. You request. Now, if you think about it, you only need to tell it about. If there are d axes, you only need to tell it about D minus one of the axes you want, because the last one it can figure out for itself right. So in total there are 10,000 by 784 numbers here altogether right.

So if you say well, I want my last axes to be 28 by 28. Then you can figure out that this must be 10,000. Otherwise it's not going to fit. It makes sense. So if you put minus 1, it says like make it as big or as small as you have to to make it fit, and so you can see here it figure it out. It has to be 10,000, so you'll see this used in neural net software, pre-processing and stuff, like that, all the time like I could have written 10,000 here, but I try to get into a habit of like anytime I'm referring to like how many items in my Input I tend to use minus one because, like it just means later on, I could like use a subsample. This code wouldn't break. I could you know, do some kind of stratified sampling. It was unbalanced. This code wouldn't break so by using this kind of approach of saying, like minus one here for the size, it just makes it more resilient to change us later. It's a good habit to get into so this kind of idea of, like being able to take tensors and reshape them and and and change axes around and stuff like. That is something you need to be like totally do without thinking, because it's going to happen all the time so, for example, here's one I tried to read in some images they were flattened. I need to unflattering them into a bunch of matrices. Okay, reshape thing. I read some: I read some images in with OpenCV and it turns out OpenCV orders the channels, blue, green red. Everything else expects them to be red, green blue.

I need to reverse the last access. How do you do that? I read in some images with place and imaging library. It reads them, as you know, rows by columns by channels quiet which expects channels by rows by columns. How do I transform that? So these are all things you need to be able to do without thinking like straightaway, because they just it happens all the time and you never want to be sitting there. Thinking about it for ages, so make sure you spend a lot of time over the week. Just practicing with things like all the stuff ago to see today, reshaping slicing reordering dimensions stuff like that, and so the best way is to create some small tensors yourself and start thinking like okay. What shall I experiment with so here? Can we pass that over there? A backtrack a little bit, of course. I love it so back in normalize. You say like you might have gone over this, but I'm still like wrestling with a little bit. Yes in many machine learning, algorithms or Milan yeah, but you also just said that scale isn't really mattered. So I said it doesn't matter for random forests yeah, so random forests are just kind of split things based on order, and so we loved them. We love random forests, further away there so immune to worrying about distributional assumptions, but we're not doing random forests. We're doing deep learning and deep learning does care.

Can you pass it over there? We have a parametric, don't we should skill? No, not quite right because, like K, nearest neighbors is nonparametric and scale matters a helluva lot. So I would say things involving trees, generally you're just going to split at a point, and so probably you don't care about scale. But you know you probably just need to think like. Is this an algorithm that uses order or does it use specific numbers, and can you please give us an intuition of why it needs scale? Just Bureau says that would make modify simulations not until we get to doing SGD, so we're going to get to that yeah. So for now we're just gon na say take my word for it: okay, positive Daniel. So this is probably a dumb question, but can you like explain a little bit more? What you mean by scale, because I guess when I think of scale I'm like? Oh all, the numbers should be generally the same size. That's like you mean, but is that, like the case like with the cats and dogs that we went over with like the deep learning like, you could have a small cat and like a larger cat, but it would still know that those were both cats. Oh, I guess you know this is one of these problems where language gets overloaded, yeah. So in computer vision, when we scale an image, we're actually increasing the size of the cat, in this case we're scaling the actual pixel values.

So, in both case, Kali means to make something bigger and smaller in this case we're taking armors from naught to 255 and making them so that they have an average of 0 and a standard deviation unit. One Jeremy, could you please explain us? Is it by column by row by pixel by pixel, so when you're scaling, I'm just not thinking about every picture, but I am kind of an input yeah, so, okay yeah sure. So I mean it's a little bit subtle, but in this case I've just got a single mean and a single standard deviation right. So it's basically on average how how much black is there right and so on average you know we have a mean and a standard deviation across all the pixels in computer vision. We would normally do it by channel, so we would normally have one number for read. One number for green one number for blue in general: you, you need a different set of normalization coefficients for each leg. Each thing you would expect to behave differently so if we were doing like a structured data set where we've got like income distance in kilometers and a number of children, but you need three separate normalization coefficients for those they're like very different kinds of things. So yeah. It's kind of like a bit domain-specific here, it's like in this case. All of the pixels are, you know, levels of gray, so we just got a single scaling number.

Where else you could imagine if they were red versus cream versus blue, you could need to scale those channels in different ways and plus they're better, please. So I'm having a little bit

of trouble imagining what would happen if you do normalize in this case um. So we'll get there so for next, oh wait, so this is kind of what your net was saying is like. Why would we normalize and for now we're normalizing, because I say we have to when we get to looking at stochastic gradient descent will basically discover that if you basically to skip ahead a little bit, we're going to be doing a matrix multiply by a bunch of Weights, we're going to pick those weights in such a way that, when we do the matrix multiply we're going to try to keep the numbers at the same scale that they started out as and that's going to basically require the initial numbers. We're going to have to know what their scale is, so basically, it's much easier to create a single kind of neural network architecture that works for lots of different kinds of inputs. If we know that they're consistently going to be mean, 0 standard deviation, 1. That would be the short answer, but we'll learn a lot more about it and if, in a couple of lessons, you're still not quite sure why let's come back to it, because it's a really interesting thing to talk about. Yes, I'm just trying to visualize the axes.

We're working with you so under plots when you, when you write x, valid shape, we get ten thousand by seven, eight four yeah that mean that we brought in 10,000 pictures yeah of that dimension, exactly okay and then in the next line. When you choose to reshape it, is there a reason why you put 28 28 on azzam Y or z, coordinates, or is there a reason why they're in that order yeah there is pretty much all neural network libraries assume that the first axis it's like kind of The equivalent of a row, it's like a separate thing, it's a sentence or an image, or you know, example of sales or whatever. So I want each image you know to be as a separate item of the first axis and then so that leaves two more axes for the rows and columns of the images and that's pretty standard. That's totally standard yeah. I don't think I've ever seen a library that doesn't work. That way. Can you pass the table here so, while normalizing the validation data, I saw you have used mean of X and standard deviation of X data training data. Only yes, so shouldn't we use mean and standard deviation of validation data you mean like join them together or separately. Goodwill didn't mean no, because you see, then you would be normalizing. The validation set using different numbers, and so now the meaning of like this. This pixel has a value of 3 in the validation set, has a different meaning to the meaning of 3 in the in the training set.

It would be like if we had like days of the week encoded such that Monday was a 1 in the training set and was a 0 in the validation set. We've got now two different sets where the same number has a different meaning. So we we want to make sure that we. So let me give an example: let's say we were doing like full-color images and our tests. Their training set can contained like green frogs, green snakes and gray elephants right we're trying to figure out which was which now we normalized using you know the each channel mean, and then we have a validation set and the test set, which are just green frogs and Green snakes, so if we would have normalized by the validation sets statistics, we would end up saying things on average a green, and so we would like remove all the greenness out, and so we would now fail to recognize the green frogs and the green snakes effectively. Right so we actually want to use the same normalization coefficients that we were training on and for those of you doing the deep learning class we actually go further than that when we use a pre tracking network, we have to use the same normalization coefficients that the Original authors trained on so the idea is that you know that the a number needs to have this consistent, meaning across every data set where you use it.

How can you pass the test meter? That means when you are looking at the test set, you normalize the test set based on this. This meanness, that's right: okay, , so um, here's a you know, so so the valid validation y-values. I just rank one tensor of 10,000. Remember, there's

this kind of weird Python thing where at Apple with just one thing in it, there's a trailing comma okay. So this is a rank, one tensor of length 10,000 and so here's an example of something from that. It's just the number three. So that's our

5. 00:34:25

- **Slicing into a tensor, 'plots()' from Fastai lib.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Labels so here's another thing: you need to be able to do in your sleep slicing into a tensor, so in this case we're slicing into the first axis with 0, so that means we're grabbing the first slice so because this is a single number, this is going To reduce the rank of the tensor by one, it's going to turn it from a 3-dimensional tensor into a two-dimensional tensor right. So you can see here. This is now just a matrix and then we're going to grab 10 through 14 inclusive rows. 10, through 14 inclusive columns - and here it is right. So this is the kind of thing you need to be super comfortable, like grabbing pieces out looking at the numbers and looking at the picture right. So here's an example of a little piece of that. First image - and so you kind of want to get used to this idea - that, if you're working with something like pictures or audio, you know this is something your brain is really good at interpreting right. So like keep showing pictures of what you're doing whenever you can. But also remember behind the scenes, they're numbers so like if something's going weird print out a few of the actual numbers you might find. Somehow some of them have become infinity or they're, all 0 or whatever right so like use this interactive environment and to explore the data as you go.

Did you have a question with just a quick? I guess semantic question: why, when it's a tensor of Rank 3, is it stored as like XYZ, instead of like tamiya, would make more sense to store it as like a list of like 2d tensors its let's do it as either right so remember the formatting, because, Let's look at this as a 3d okay, so here's a 3d right, so a 3d tensor is formatted as showing a list of 2d tensors, basically, but when you're extracting it, why isn't it like if you're extracting the first one? Why isn't it X images square brackets? 0, close square brackets and then a second set of squares, because this has a different meaning right. So it's kind of the difference between tenses and jagged arrays right. So basically, if you do like something like something like that, that says take the second list item and from it grab the third list item. And so we tend to use that when we have something called a jagged array, which is where each sub array may be of a different length right. Where else we have like a single object of three dimensions and so we're trying to say like which little piece of it do we want, and so the idea is that that is a single slice object to go in and grab that piece out. Ok, so here's an example of a few of those images along with their labels and this kind of stuff. You want to be able to do pretty quickly with matplotlib.

It's it's going to help you a lot in in life in your exam, so you can have a look at you know what Rachel wrote here when she wrote plots we can use, we can use, add sub plot to basically create those little separate plots and you Need to know that I am show is how we basically take a numpy array and draw it as a picture. Okay and then we've also added the title on top, so there it is all right. So, let's you know,

6. 00:38:20

- **Overview of a Neural Network**
- **Michael Nielsen universal approximation theorem: a visual proof that neural nets can compute any function**
- **Why you should blog (by Rachel Thomas)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Take that data and try to build a neural network with it and so a neural network and sorry this is going to be a lot of review. For those of you already doing deep learning, a neural network is just a particular mathematical function or a class of mathematical functions, but it's a really important class because it has the property it supports. What's called the universal approximation theorem, which is that which means that a neural network can approximate any other function arbitrarily closely right, so in other words, it can do in theory. It can do anything as long as we make it big enough. So this is very different to a function like 3x plus 5 right, which can only do one thing. It's a very specific function or the class of functions ax plus B, which can only represent lines of different slopes, moving it up and down different amounts, or even the function. Ax, squared plus BX, plus C plus sine D. You know again only can represent a very specific subset of relationships. The neural network, however, is a function that can represent any other function to arbitrarily close accuracy right. So what we're going to do is we're going to learn how to take a function. So, let's take like ax plus B and we're going to learn how to find its parameters, in this case a and B which allow it to fit as closely as possible to a set of data.

And so this here is showing example from a notebook that we'll be looking at in deep learning course, which basically shows what happens when we use something called stochastic gradient descent to try and set a and B and basically what happens is we're going to pick a Random, a to start with a random B to start with, and then we're going to basically figure out. Do I need to increase or decrease a to make it closer at the line closer to the dots? Do I need to increase or decrease B to make the line closer to the dots and then just keep increasing and decreasing a and B lots and lots of times? Okay? So that's what we're going to do and to answer the question: do I need to increase or decrease a and B we're going to take the derivative right, so the derivative of the function with respect to a and B tells us: how will that function? Change as we change a and B all right, so that's basically what we're going to do, but we're not going to start with just a line. The idea is we're to build up to actually having a neural net, and so it's going to be exactly the same idea, but because it's an infinitely flexible function, we're going to be able to use this exact same technique to fit arbitrarily to arbitrarily complex relationships. Now, that's basically the idea. So then what you need to know is that a neural net is actually a very simple thing. A neural net actually is something which takes as input.

Let's say, we've got a vector does a matrix product by that vector right. So this is like this is of size. Let's draw this properly so like if this is sighs. Ah, this is like our bye see a matrix product will spit out something of size C all right, and then we do something called a non-linearity, which is basically we're going to throw away all the negative values. So can so it's basically max zero, comma X and then we're going to put that through another matrix multiply and then we're going to put that through another max zero comma X and we're going to put that through another matrix multiply and so on. Right until eventually, we end up with the single vector that we want, so, in other words, each stage of our neural network is the key thing going on is a matrix multiplied so, in other words, a linear function. So, basically, deep

learning most of their calculation is lots and lots of linear functions, but between each one we're going to replace the negative numbers with zeros. Can you yes, so why are we throwing away the negative numbers? Well, we'll see right. The short answer is: if you apply a linear function to a linear function to a linear function, it's still just a linear function, so it's totally useless. But if you throw away the negatives, that's actually a nonlinear transformation, and so it turns out that if you apply a linear function to the thing we threw away the negatives.

That applies that to a linear function. That creates a neural network and it turns out. That's the thing that can approximate any other function arbitrarily closely, so this tiny little difference actually makes all the difference and if you're interested in it check out the deep learning video where we cover this, because I actually show a nice visual intuitive proof, not something that I created that's something that Michael Nielsen created or if you want to skip straight to his website, you could go to Michael Nielsen universe, or I think I spelled his name wrong, never mind Haitian theorem, . We go neural networks and deep learning, chapter four and he's got a really nice walkthrough, basically with lots of animations, where you can see why this works one. I feel like the the hardest thing. I feel like the hardest thing with getting started like technical writing on the Internet is just like posting, your first thing, so, if you do a search for Rachel, Thomas medium blog you'll find, this will put it on the lesson wiki, where she talks about. She actually says the top advice she would give to her. Younger self would be to start blogging sooner, and she has like both reasons why you should do it some examples of things that you know examples of places. She's blog has turned out to be great for her and her career, but then some tips about how to get started.

I remember when I first suggested to Rachel she might think about blogging, because she had so much interesting to say, and you know at first he was kind of surprised at the idea that, like she could blog you know, and now people come up to us at Conferences and they're, like you're, Rachel Thomas, I love your writing. You know so, like I've kind of seen that that transition from like wow could I blog to being known as a strong technical author, so yeah so check out this article. If you still need convincing or if you're wondering how to get started and since the first one is the hardest, maybe your first one should be like something really easy for you to write. You know so it could be. Like you know, here's a summary of the first 15 minutes of lesson. Three of our machine learning costs. You know here's why it's interesting, here's what we lit, or it could be like here's, a summary of how I used to random forests to solve a particular problem. In my practicum I often get questions like on my practicum, my organization. We've got like sensitive commercial data. That's fine, like you know, just find another data set and do it on that stair to show the example or you know, anonymize all of the values and change the names of the variables or whatever, like you, can talk to your employer or your practicum partner to Make sure that they're comfortable with whatever it is you're writing in general, though you know, people love it when they're interns and staff blog about what they're working on, because it makes them look super cool.

You know it's like hey, I'm on. You know intern working at this company and I wrote this post about this cool analysis. I did and then other people would be like wow. That looks like a great company to work for so generally speaking, you should find people are pretty supportive um, besides, which there's lots and lots of data sets out there available. So, even if you can't base it on the work you're doing, you can find something similar for sure. Alright,

7. 00:47:15

- **Intro to PyTorch & Nvidia GPUs for Deep Learning**

- **Website to buy a laptop with a good GPU: xoticpc.com**
- **Using cloud services like Crestle.com or AWS (and how to gain access EC2 w/ "Request limit increase")**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So we're going to start building our neural network, we're going to build it using something called a torch. pytorch is a library that basically looks a lot like numpy, but when you create some code with pytorch, you can run it on the GPU rather than the CPU. So the GPU is something which is basically going to be probably at least an order of magnitude, possibly hundreds of times faster than the code that you might write for the CPU for particularly stuff involving lots of linear algebra right so with deep learning neural nets. You can, if you, if you don't, have a GPU, you can do it on the CPU right, but it's it's going to be frustratingly slow. Your Mac does not have a GPU that we can use for this because I'm actually talking today, we need an NVIDIA GPU. I would actually much prefer that we could use your Mac's, because competition is great right, but in video, we're really the first ones to create a GPU which did a good job of supporting general purpose: graphics, programming, units GPGPU. So, in other words, that means using a GPU for things other than playing computer games. They used, they created a framework called CUDA, see UDA. It's it's a very good framework. It's pretty much universally used in deep learning. If you don't have an NVIDIA GPU, you can't use it. No current max have an NVIDIA GPU, most laptops of any kind. Don't have an NVIDIA GPU. If you're interested in doing deep learning on your laptop.

The good news is that you need to buy one which is really good for playing computer games on there's a place called exotic PC gaming laptops where you can go and buy yourself a great lap up for doing deep learning. You can tell your parents that you need the money to do deep learning, so could you please have yeah so you'll generally find a whole bunch of laptops with names like predator and Viper, with pictures of robots and stuff, so stealth, pro Raider leopard anyway. Having said that, like, I don't know that many people that do much deep learning on their laptop most people will log into a cloud environment. By far the easiest. I know a few users called cresol with Crestle. You can basically sign up and straightaway. The first thing you get is a throne straight into a jupiter notebook, backed by a GPU cost 60 cents an hour with all of the fastai libraries and data already available. So that makes life really easy. It's less flexible and in some ways less fast, then using AWS, which is the Amazon Web Services, option, cost a little bit more ninety cents, an hour rather than 60 cents, and but it's very likely that your employer is already using that. It's like it's good to get to know anyway. They've got more different choices around GPUs and it's a good good choice. If you're, Google for github student pack, if you're a student, you can get a hundred and fifty dollars of credits straight away pretty much, and so it's a really good way to get started Daniel. Did you have a question yeah? I just wanted to know your opinion on.

I know that until recently published like an open source like way of like boosting like regular packages, that they claim is equivalent like, if you use the bottom tier GPU on your seat, like on your CPU. If you use their boost packages like you, can get the same performance, do you know anything about that yeah? I do it's a good question, so I'm actually intro makes some great numerical programming libraries, particularly this one called mkl, the matrix, colonel library. They they definitely make things faster, they're, not using those libraries. But if you look at a graph of performance over time, GPUs have consistently throughout the last 10 years, including now are about 10 times more floating-point operations per second than the

equivalent CPU and they're generally about a fifth of the price for that performance. So yeah it and then because of that, like everybody doing anything with deep learning, basically does it on NVIDIA GPUs and therefore using anything other than NVIDIA GPUs is currently very annoying, so slower more expensive more annoying. I really hope there will be more activity around AMD. Gpus, in particular in this area, but AMD's got like literally years of catching up to do so it might take a while yeah. So I just wanted to point out that you can also buy at things such as, like a GPU extender to a laptop yeah.

That's also like kind of making, like maybe a first step solution. If you only really want to yeah yeah, I think for like 300 bucks, or so you can buy something that plugs into your Thunderbolt port. If you have a Mac and then for another five or six hundred bucks, you can buy a GPU to plug into that. Having said that, for about a thousand bucks, you can actually create a pretty good. You know GPU based desktop, and so, if you're, considering that the fastai forums have like lots of threads, where people help each other spec out something at a particular price point anyway. So to start with out so use cresol, and then you know, when you're ready to invest a few extra minutes getting going use AWS to use AWS you basically huh yeah yeah yeah, I'm just talking to the folks online as well. Okay, so so AWS! When you get there, go to ec2 ac2, like there's lots of stuff on AWS ec2 is the bit where we get to like a rent computers by the hour right now. We're gon na need a GPU based instance. Unfortunately, when you first sign up for AWS, they don't give you access to them, so you have to request that access so go to limits up in the top left right and the main GPU instance we'll be using is called the p2 so scroll down to p2 And here P, 2 dot X large. You need to make sure that that numbers not 0. If you've just got a new account, it probably is 0, which means you won't be allowed to create one.

You have to go request, limit increase and the trick there is when it asks you. Why do you want the limit increase, type faster, AI, because AWS knows to look out and they know that faster day I people are good people so they'll. Do it quite quickly that takes a day or two generally speaking, to go through so once you get the email saying you've been approved for p2 instances, you can then go back here and say launch instance, and so we've basically set up one. That has everything you need. So if you click on community, ama and ami is an Amazon machine image, it's basically a completely set up computer right, and so, if you type fastai or one word, you'll find here fastai do part one version two for the p2 right. So that's all set up ready to go. So if you click on select and it'll say: okay, what kind of computer do you want right, and so we have to say all right. I want a GPU compute type and specifically I want to pee to extra-large right and then you can say, review and launch. I'm assuming you already know how to deal with SSH keys and all that kind of stuff. If you don't check out the introductory tutorials and workshop videos that we have online or Google around for SSH keys, very important skill to know anyway, all right, so hopefully you get through all that you have something running on a GPU with the past AI repo.

If you use cresol, just CD fastai to the the repo is already there get Paul AWS CD, fastai, the repo is already there get Paul. If it's your own computer you'll just have to get clone and then away. You go all right. So part of all of those is PI, torch is pre-installed and so pytorch basically means we can write code. That looks a lot like numpy, but it's going to run really quickly on the GPU. Secondly, since we need to know like which direction and how much to move our parameters to improve our loss, we need to know the derivative of functions. pytorch. Has this amazing thing where any code you write using the pytorch library, it can automatically take the derivative of that for you. So we're not going to look at any capitalist in this course, and I don't look at any calculus in any of my courses or at any of my work,

basically ever in terms of like actually calculating derivatives myself, because I've never had to it's done. For me by the library, so as long as you write the Python code, it's the derivative, it's done so the only calculus you really need to know to be an effective practitioner is like what is it? What does it mean to be a derivative, and you also need to know the chain rule which will come all right, so we're going to

8. 00:57:45

- **Create a Neural Net for Logistic Regression in PyTorch**
- **'net = nn.Sequential(nn.Linear(28*28, 10), nn.LogSoftmax()).cuda()'**
- **'md = ImageClassifierData.from_arrays(path, (x,y), (x_valid, y_valid))'**
- **Loss function such as 'nn.NLLLoss()' or Negative Log Likelihood Loss or Cross-Entropy (binary or categorical)**
- **Looking at Loss with Excel**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Start out kind of top-down, create a neural net and we're going to assume a whole bunch of stuff and gradually we're going to dig into each piece right so to create neural Nets. We need to import the pytorch, neural net library, pytorch. Funnily enough is not called play torch, it's called torch, okay, so torch NN is the pytorch subsection, that's responsible for neural nets. Okay! So we'll call that n N and then we're going to import a few bits out of fastai, just to make life a bit easier for us. So here's how you create a neural network, n, pytorch by the simplest possible neural network. You say, sequential and sequential means I am now going to give you a list of the layers that I want in my neural network right. So in this case my list has two things in it. The first thing says I want a linear layer. Now a linear layer is something that's basically going to do. Y equals ax, plus B right but matrix matrix multiply, not not univariate. Obviously, so it's going to do a matrix product. Basically, so the input of the matrix product is going to be a vector of length. 28 times 28, because that's how many pixels we have and the output needs to be of size 10, we'll talk about why in a moment, but for now you know this is how we define a linear layer and then again we're going to dig into this in Detail, but every linear layer, just about in neural nets, has to have a non-linearity after it then we're going to learn about this particular non-linearity. In a moment it's called the softmax and if you've done, the DL course you've already seen this.

So that's how we define a neuron it. This is a two layer: neural net there's also kind of an implicit additional first layer, which is the input, but with pytorch you don't have to explicitly mention the input that normally, we think conceptually like the input. Image is kind of also a layer, because we're kind of doing things pretty manually with pytorch, we're not taking advantage of any of their convenience, is in fastai. For building your stuff, we have to then write CUDA, which tells pytorch to copy this neural network across to the GPU. So now, on from now on, that network is going to be actually running on the GPU. We didn't say that it would run on the CPU, so that gives us back a neuron it very simple, neural net. So we're then going to try and fit the neural net to some data. So we need some data. So fastai has this concept of a model data object, which is basically something that wraps up: training, data, validation, data and optionally test data, and so to create a model data object. You can just say I want to create some image. Classifier data I'm going to grab it from some arrays right and you just say: ok, this is the path. Then I'm going to save any temporary files. This is my training data arrays, and this is my validation, data, arrays, ok, and so that just returns an

object. That's going to wrap that all up and so we're going to be able to fit to that data.

So now that we have a neural net and we have some data - we're going to come back to this in a moment, but we basically say what loss function do we want to use what optimizer do we want to use and then we say fit, we say Fit this network to this data going over every image once using this loss function this optimizer and print out these metrics bang. Ok - and this says here - this is 91.8 % accurate. Ok, so that's like the simplest possible neuron it. So what that's doing is it's? Creating a matrix multiplication followed by a non-linearity and that it's trying to find the values for this matrix, which cause which basically that fit the data as well as possible that a product that end up predicting this is a 1. This is a 9s is a 3, and so we need some definition for as well as possible, and so the general term, for that thing is called the loss function. So the loss function is the function. That's going to be lower. If this is better right. Just like with random forests, we had this concept of information gain and we got to like pick what function do you want to use to define information gain and we were mainly looking at root, mean square error right, most machine learning, algorithms recall something very similar to At loss right, so the loss is: how do we score how good we are, and so, in the end, we're going to calculate the derivative of the loss with respect to the weight matrix that we're multiplying by to figure out how to how to update it right? So we're going to use something called negative log likelihood loss so negative log likelihood.

Loss is also known as cross entropy, they're literally the same thing, there's two versions: one called binary cross, entropy or binary negative log likelihood and another court categorical cross. Entropy the same thing. One is for when you've only got a zero or one dependent, the other is if you've got like cat dog airplane or horse, or at 0, 1 through 9 and so forth. So what we've got here is the binary version of cross entropy, and so here is the definition. I think maybe the easiest way to understand this definition is to look at an example. So let's say we're trying to protect cat versus dog. One is cat. Zero is dog, so here we've got cat dog dog cat and here are our predictions. We said 90 % sure it's a cat, 90 % sure it's a dog, 80 % sure it's a dog, 80 % sure it's a cat all right. So we can then calculate the binary cross entropy by calling our function. So it's going to say: okay for the first one, we've got y equals 1. So it's going to be 1 times log of 0.9 plus 1 minus y 1. Minus 1 is 0. So that's going to be skipped, okay and then the second one is going to be a 0. So it's going to be 0 times something, so that's going to be skipped and the second part will be 1 minus 0. Ah, so this is 1 times log of 1 minus p1. Minus point. 1 is point 9, so in other words, the first piece and the second piece of this are going to give exactly the same number, which makes sense because the first one we said we were 90 %, confident it was a cat and it was and the second We said we were 90 %, confident it was a dog and it was so in each case. The loss is coming from the fact that you know we could have been more confident yeah.

So if we said we're a hundred percent confident the loss would have been zero right. So let's look at that in Excel. So here's our Oh point, nine point. One point two point: eight right and here's our predictions 101. So here's one minus the prediction right here is log of our prediction: here is log of one minus our prediction, and so then here is our sum. Okay, so if you think about it - and I want you to think about this during the week - you could replace this with an if statement rather than Y, because Y is always 1 or zero right. Then it's only ever going to use either this or this. So you could replace this with an if statement. So I'd like you during the week to try to rewrite this with an if statement, okay and then see if you can then scale it out to be a categorical cross-entropy, so categorical cross-entropy works this way. Let's say we were trying to predict three and then six and then seven and then two so if we were trying to predict three and the actual thing that was predicted was like four point: seven

right versus like well actually putting it. This way we're trying predict three and we actually predicted five or a try to predict three, and we accidentally predicted. Nine like Bing. Five, instead of three is no better than being mine instead of three. So we're not actually going to say like how far away is the actual number we're going to express it differently or to put it another way.

What, if we're trying to predict cats, dogs, horses and airplanes, you can't like how far away is cat from horse, so we're going to express these a little bit differently rather than thinking of it as a three, let's think of it as a vector with a 1. In the third location, and rather than thinking it was a six, let's think of it as a vector of zeros for the one in the sixth location, so in other words one hot encoding right. So that's one hot in code, a dependent variable, and so that way now, rather than predicting trying to predict a single number, let's predict ten numbers. Alright, let's predict, what's the probability that it's a zero? What's the probability it's a 1: what's the probability, it's a 2 and so forth, alright, and so let's say we're trying to predict the two right. Then here is our binary: cross-entropy, sorry, categorical cross-entropy! So it's just saying: okay did this one predict correctly or not? How far off was it and so forth for each one all right and so add them all up so categorical cross-entropy is identical to binary cross-entropy. We just have to add it up across all of the categories, so try and turn the binary cross-entropy function in python into a categorical, cross-entropy python and maybe create both the version with the if statement and the version with the sum and the product, alright. Alright. So that's why, in our pytorch, we had 10 as the output as the put dimensionality for this matrix, because when we multiply sup by a matrix with 10 columns, we're going to end up with something of length 10, which is what we want.

We want to have 10 predictions, ok, so so that's the loss function that we're using all right. So

9. 01:09:05

- **Let's fit the model then make predictions on Validation set.**
- **'fit(net, md, epochs=1, crit=loss, opt=opt, metrics=metrics)'**
- **Note: PyTorch doesn't use the word "loss" but the word "criterion", thus 'crit=loss'**
- **'preds = predict(net, md.val_dl)'**
- **'preds.shape' -> (10000, 10)**
- **'preds.argmax(axis=1)[:5]', argmax will return the index of the value which is the number itself.**
- **'np.mean(preds == y_valid)' to check how accurate the model is on Validation set.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Then we can fit the model and what it does is. It goes through every image this many times. So, in this case, it's just looking at every image once and go into slightly update the values in that weight, matrix based on those gradients, and so once we've trained it. We can then say predict using this model on the validation set right and now that spits out something of 10,000 by 10 consumers Elmi. Why is this shape these predictions? Why are they have shaked 10,000 by 10? Don't fret Chris, it's right next to you! Well, it's because we have 10,000 images, we're training up 10,000 images training on. So that's what we're validating on north coast, but it's the same thing so 10,000 be validating on. So that's the first axis as percent. The second axis is because we actually make 10 predictions per image good good exactly so. Each one of these

rows is the probabilities that it's Anor, that it's a one that isn't true, that's three and so forth: okay, very good! So in math there's a really common operation. We do called Arg max notice, though it's common, it's funny you like at high school. I never saw odd max first year undergrad. I never saw Arg max, but somehow after university everything's about Arg max. So one of these things - that's for some reason not really taught at school, but it actually turns out to be super critical and so odd.

Max is both something that you'll see in math and it's just written out in full Arg max it's in numpy. It's in pytorch, it's super important and what it does is it says: let's take this array of creds right and let's figure out on this axis. Remember access one is columns right so across, as Chris said, the 10 predictions for each one for each row. Let's find which prediction has the highest value and return, not that if it does hit max it would return the value. Arg max returns the index with their value, all right. So by saying arc max Axess equals 1. It's going to return the index, which is actually the number itself right. So let's grab the first 5 okay, so for the first one it thinks is a 3. Then it thinks six one's an eight next one's a six, the next one's, a nine next one's a six again. Okay. So that's how we can convert our probabilities back into predictions all right. So if we I'm safe that away call it Preds. We can then say: okay when does Preds equal the ground truth right. So that's going to return an array of balls which we can treat as ones and zeros and the mean of a bunch of ones and zeroes is just the average. So that gives us the accuracy so there's our 91.8 %, and so you want to be able to like replicate the numbers you see, and here it is there's our 91.8 % all right. So when we train this it tells us.

The last thing it tells us is whatever metric we asked for and we asked for accuracy, okay, so the last thing it tells us is our metric, which is accuracy and then, before that we get the training set loss and the loss is again whatever odds we Asked for negative log likelihood, and the second thing is the validation set loss. pytorch doesn't use the word loss, they use the word criterion, so you'll see here, crit, okay, so that's criterion equals loss. This is what loss function. Do we want to use? They call that the criterion same thing, okay, so here's how we can recreate that accuracy. So now we can go ahead and plot eight of the images along with their predictions and we've got three eight six: nine, Oh wrong, five wrong! Okay - and you can see like why they are wrong, like this - is pretty close to a nine. It's just missing a little cross at the top. This is pretty close to a five. It's got a little bit at the extra here right, so we've made a start and, and all we've done so far is we haven't actually created a deep neural net. We've actually got only one layer, so what we've actually done is we've created a logistic regression. Okay, so a logistic regression is is literally what we just built and you could try and replicate this with scikit-learn logistic regression package.

When I did it, I got similar accuracy, but this version ran much faster because this is running on the GPU. Where else scikit-learn runs on the CPU okay, so even for something like logistic regression, we can, you know, implement it very quickly, with pytorch, actually pass that in so when we're. When we're creating our net, we have to do dot CUDA. What would be the consequence of not doing that? What it does not run? It wouldn't run quickly, yeah it'll run on the CPU. Can you pass it to jail, so may build up your network? Why is that? We have to do linear and followed by a nonlinear. So the short answer is because that's what the universal approximation theorem says is the structure which can give you arbitrarily accurate functions for any functional form. You know, so the long answer is the details of why the universal approximation theorem works. Another version of the short answer is that's the definition of a neural network, so the definition of a neural network is a linear layer, followed by a activation function, followed by a linear layer, followed by an activation function, etc. We go

into a lot more detail of this in the deep learning, of course, but you know for this purpose: it's it's enough to know like that. It works so far. Of course, we haven't actually built a deep neural net at all.

We've just built a logistic regression, and so at this point, if you think about it, all we're doing is we're taking every input, pixel and multiplying it by a weight for each possible outcome right. So we're basically saying you know on average the number one you know has these pixels turned on the number two has these pixels turned on and that's why it's not terribly accurate right. That's that's! Not how digit recognition works in real life, but that's that's always built. So far, ok, can you pass that to Devon, so you

10. 01:16:05

- **A second pass on "Michael Nielsen universal approximation theorem"**
- **A Neural Network can approximate any other function to close accuracy, as long as it's large enough.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Keep saying this Universal approximation theorem yeah. Did you define yeah, but let's cover it again, because it's worth talking about so all right. So Michael Nielsen has this great website called neural networks and deep learning and his chapter 4 is actually kind of famous now and in it he does this walkthrough of basically showing that a neural network can can approximate any other function to arbitrarily close accuracy. As long as it's big enough - and we walk through this in a lot of detail in the deep learning course, but the basic trick is that he shows that, with a few different numbers, you can basically kind of cause these things to kind of create little boxes. You can move the boxes up and down. You can move them around. You can join them together to eventually basically create like connections of towers, which you can like use to approximate any kind of surface right. So, that's you know, that's basically the trick, and so all we need to do given given that is to kind of find the parameters for each of the linear functions in that neural network. So to find the weights in each of the in each of the matrices, and so so far we've got just one matrix and so we've just built a simple, logistic regression. So far, just a small doubt. I just want to confirm that when you showed images of the examples of the images which were misclassified yeah, they look rectangular. So it's just that, while rendering pixels are being scale differently, so are they still 28 by 28 Square 20? Hyper 20? I think it's square. I think I just look like angular cuz they've got titles on the top, I'm not sure yeah.

I don't know anyway, they are square and um like matplotlib yeah

11. 01:18:15

- **Defining Logistic Regression ourselves, from scratch, not using PyTorch 'nn.Sequential()'**
- **Demo explanation with drawings by Jeremy.**
- **Look at Excel 'entropy_example.xlsx' for Softmax and Sigmoid**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

It does often fiddle around with you know what it considers black versus white and you know having different size, axes and stuff. So yeah you do have to be careful there, sometimes um, okay. So hopefully this will now make more sense, because what we're going to do is like dig in a layer deeper and define logistic regression without using an n dot sequential without using an end on linear without using an end log softmax. So we're going to do nearly all of the layer definition from scratch. Ok! So, to do that, we're going to have to define a pytorch module, a pytorch module is basically either a neural net or a layer in a neural net which is actually kind of a powerful concept of itself. Basically, anything that can kind of behave like a neural net can itself be part of another neuron net, and so this is like how we can construct particularly powerful architectures, combining lots of other pieces so to create a pytorch module just create a Python class, but It has to inherit from an end module, so we haven't done inheritance before other than that. This is all the same concepts we've seen an oo already. Basically, if you put something in parentheses here, what it means is that our class gets all of the functionality of this class for free. It's called sub classing it. So we're going to get all of the capabilities of a neural network module that the pytorch authors have provided and then we're going to add additional functionality to it.

When you create a sub class, there is one key thing you need to remember to do, which is when you initialize your class. You have to first of all initialize the superclass right, so the superclass is the NN module, so the NN module has to be built before you can start adding your pieces to it, and so this is just like something you can copy and paste into every one Of your modules, you just say super dot in it. This just means construct the superclass first, ok, so having done that, we can now go ahead and define our weights and our bias, so our weights is the weight. Matrix is the actual matrix that we're going to multiply our data by and as we discuss it's going to have 28 times, 28 rows and 10 columns. And that's because if we take an image which we flattened out into a 28 by 28 length vector right, then we can multiply it by this weight matrix to get back out a length 10 vector, which we can then use to consider it as a set of Predictions so that's our weight matrix now. The problem is that we don't just want y equals ax. We want y equals ax plus B, so the plus B in neural nets is called bias, and so as well as defining weights, we're also going to find bias, and so since this thing is going to spit out for every image something of length 10, that means That we need to create a vector of length 10 to be our biases, in other words, for everything naught 1 2 3 up to mine, we're going to have a different plus B. That would be adding right. So we've got our data matrix here, which is of length 10,000 by 28 times 28, all right and then we've got our weight matrix, which is 28 by 28 rose by 10. So if we multiply those together, we get something of size 10,000 by 10, okay and then we want to add on our bias up run way around add on our bias.

Okay, like so, and so when we add on and we're going to learn a lot more about this later. But when we add on a vector like this, it basically is going to get added to every row. Okay, so that bias is going to get added to every road, so we first of all define those and so to define them. We've created a tiny little function called get weights, which is over here, alright, which basically just creates some normally distributed random numbers. So a torch Rand n returns a tensor filled with random numbers from a normal distribution. We have to be a bit careful, though, when we do deep learning like when we add more linear layers later imagine if we have a matrix, which, on average tends to increase the size of the inputs we give to it. If we then multiply it by lots of matrices of that size, it's going to make the numbers bigger and bigger and bigger like exponentially bigger. Well, what if it made them a bit smaller, it's going to make them smaller and smaller and smaller exponentially smaller, so like, because a deep network applies lots of linear layers if, on average, they result in things a bit bigger than they started with or a bit Smaller than they started with it's going to like exponentially multiply that

difference, so we need to make sure that the weight matrix is of an appropriate size that the inputs to it, they're kind of the mean of the inputs, basically is not going to change.

So it turns out that if you use normally distributed random numbers and divided by the number of rows in the weight matrix, it turns out that particular random initialization keeps your numbers at about the right scale right. So this idea that, like if you've done linear algebra, basically if the eigen value the first eigenvalue, is like bigger than one or smaller than one, it's cause the gradients to like get bigger and bigger or smaller and smaller. That's called gradient explosion right. So we'll talk more about this in the deep learning course, but if you're interested you can look at climbing her initialization and read all about this concept right. But for now you know it's probably just enough to know that if you use this type of random number generation, you're gon na get random numbers that are and nicely behaved, you're going to start out with an input which is mean 0 standard deviation. 1. Once you put it through this set of random numbers, you'll still have something: that's about mean. 0 standard deviation. 1, that's! Basically the goal. Okay, one nice thing about pytorch is that you can play with this stuff right, so choice, dot, Randi and like try it out. Every time you see a function being used, run it go ahead and take a look and so you'll see. It looks a lot like numpy right, but it doesn't return a numpy array. It returns a tensor and in fact now I'm GPU programming, okay, like put CUDA, and now it's doing it on the GPU so like I just multiplied that matrix by 3 very quickly on the GPU right.

So that's how we do GPU programming with pay torch. All right, so this this is our weight matrix, so we create, as I said, we create one pretty at the 28 by 10. 1 is just Rank 1 of 10 for the biases. We have to make them a parameter. This is basically telling pytorch, which things to update when it does SGD, that's very minor technical detail. So, having created the weight matrices, we then define a special method with the name forward. This is a special method. The word the name forward has a special meaning in pytorch. A method called forward in pytorch is the name of the method that will get called when your layer is calculated. Ok, so if you create a neural net or a layer, you have to define forward and it's going to get past the data from the previous layer. So our definition is to do a matrix multiplication of our input data times our weights and add on the biases. So that's it that's what happened earlier on when we said n n dot, linear that created this this thing for us! Okay! Now, unfortunately, though, we're not getting a 28 by 28 long vector we're getting a 28 row by 28 column matrix, so we have to flatten it. Unfortunately, in torch high torch, they tend to rename things they they spell re sigh reshape. They spell it view. Okay, so view means reshape. So you can see here we end up with something where the number of images we're going to leave the same and then we're going to replace row by column with a single axis again negative one meaning as long as required.

Okay. So this is how we flatten something using pytorch, so we flatten it do a matrix multiply and then finally, we do our soft maps. So softmax is the activation function we use. If you look in the deep learning repo you'll find something called entropy example, where you'll see an example of softmax, but a soft Mac simply takes the outputs from our final layer. So we get our outputs from our from our linear layer and what we do is we go e ^ for each output and then we take that number and we divide by the sum of the eight of the perils that's called softmax. Why do we do that? Well, because we're dividing this by the sum that means that the sum of those itself must add to one right and that's what we want, we want the probabilities of all the possible outcomes. Add to one, furthermore, because we're using e ^. That means we know that everyone, these is between zero and one and probabilities we know, should be between zero and one and then, finally, because we're using e to the power of it, tends to

mean that slightly bigger values in the input turn into much bigger values. In the output so you'll see, generally speaking my softmax, there was going to be one big number and lots of small numbers and that's what we want right, because we know that the output is one hot encoded, so in other words a softmax activation function.

The softmax non-linearity is something that returns things that behave like probabilities and where one of those probabilities is more likely to be kind of high and the other ones are more likely to be low. And we know that's what we want for a to map to our one hot encoding. So a softmax is a great activation function to use to kind of help. The neural net make it easier for the neural net to to map to the output that you wanted, and this is what we generally want. When we're kind of designing neural networks, we try to come up with little architectural tweaks that make it as easy for it. As possible to to to match the output that we know we want, so that's basically it right like, rather than doing sequential, you know and using an end, linear and in dot softmax we've defined it from scratch. We can now say just like before our net is equal to that plastic CUDA and we can say, dot fetch and we get to within a slight random deviation, exactly the same output.

12. 01:31:05

- **Assignements for the week, student question on 'Forward(self, x)'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, so I'm what I'd like you to do during the week is to play around with, like torture and n to generate some random tensors Torche drop map mole to start multiplying them together. Adding them up try to make sure that you can rewrite softmax yourself from scratch. You know like try to fiddle around a bit with you know reshaping view or that kind of stuff, so that by the time you come back next week, you feel like pretty comfortable with pytorch, and if you, google, for pytorch tutorial, you'll see there's a Lot of great material actually on the pytorch website, to help you along basically showing you how to create tensors and modify them and do operations on them all right, great. Yes, you had a question: can you pass it over? So I see that the forward is the layer that gets applied after each of the linear layers. So not quite the forward is just the definition of the module, so this is like how it this is how we're implementing linear. So does that mean after each linear layer you have to apply the same function. Let's say we can't do a log softmax after layer. One and then apply some other function after layer two. If we have like a multi-layer neural network, so normally way to find neural networks. Normally we define neural networks like so we just say here is a list of the layers we wander right. We don't you, don't have to write your own forward right.

All we did just now is to say like okay, instead of doing this, let's not use any of this at all, but write it all by hand ourselves all right. So you can. You can write as many layers you see like in what any order you like here. The point was that here we're not using any of that we've written our own mat mole plus bias our own softmax. So this is like this is. This is just Python code. You can write whatever Python code inside forward that you like to define your own neural net. So, like you, you won't normally do this yourself, normally you'll just use the layers that pie chart provides and your use dot, sequential to put them together or even more likely. Your download or predefined architecture and use that we're just doing this to learn how it works behind the scenes all right, great thanks. Everybody

Lesson 09

Outline

- Deep Learning
- Using pytorch and a 1-level NN
- Walkthrough of MNIST number sets
- Binary Loss func
- Making a LogReg equivalent NN pytorch

Video Timelines and Transcript

Jeremy starts with a selection of students' posts.

1. 00:00:01

- **Structuring the Unstructured: a visual demo of Bagging with Random Forests.**
- **http://structuringtheunstructured.blogspot.se/2017/11/coloring-with-random-forests.html**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

All right welcome back to machine learning. I am really excited to be able to share some amazing stuff that University of San Francisco students have built during the week or written about during the week, and quite a few things are going to show. You have already spread around the internet quite a bit: lots of tweets and posts and all kinds of stuff happening, one of the the first to be widely shared. Was this one by Tyler? Who did something really interesting? He he started out by saying like what. If I like, create the synthetic data set where the independent variables is like the X and the y and the dependent variable is like color right and interestingly, he showed me an earlier version of this, where he wasn't using color. He was just like putting the actual numbers in here and this thing kind of wasn't really working at all and as soon as he started using Kawa, it started working really well, and so I wanted to mention that one of the things that unfortunately we we don't Teach you at USF is a theory of human perception. Perhaps we should, because actually, when it comes to visualization, its kind of the most important thing to know is what is the human eye or what is what what it was the human brain good at perceiving? There's a whole area of academic study on this, and one of the things that we're best at perceiving is differences in color right.

So that's why, as soon as we look at this picture of this synthetic data, he created, you can immediately say. Oh there's kind of four areas of you know lighter red color, so what he did was he said: okay, what if we like tried to create a machine learning model of this synthetic data set, and so specifically, he created a tree, and the cool thing is that You can actually draw the tree right so after he created the tree. He did this all in matplotlib. Matplotlib is very flexible right. He actually drew the tree boundaries. So that's already. A pretty neat trick is to

be actually able to draw the tree, but then he did something even cleverer, which is he said, okay. So what predictions does the tree make well as the average of each of these areas, and so to do that we can actually draw the average color. Alright, there's actually kind of pretty here is the predictions that the tree makes now. Here's where it gets really interesting is like you can, as you know, randomly generate trees through resampling, and so here are four trees generated through resampling they're, all like pretty similar, but a little bit different, and so now we can actually visualize bagging and to visualize bagging. We literally take the average of the four pictures, all right, that's what bagging is and there it is alright, and so here is like the the fuzzy decision boundaries of a random forest, and I think this is kind of amazing right, because it's it's like a.

I wish I had this actually when I started teaching you all random forest, because I could have skipped a couple of classes. It's just like. Okay, that's what we do. You know we create the decision, boundaries, we average each area and then we we do it. A few times in average, all of them, okay. So that's what a random forest does, and I think like this is just such a great example of making the complex easy through through pictures so congrats to Tyler, for that it actually turns out that he has actually reinvented something that somebody else has already done. A guy called Christian any who went on to be one of the world's foremost machine learning. Researchers actually included almost exactly this technique. In a book he wrote about decision forests, so it's actually kind of cool that Tyler ended up reinventing something that one of the world's foremost authorities on v decision forests. Actually it has created. So I

2. 00:04:01

- **Parfit: a library for quick and powerful hyper-parameter optimization with visualizations.**
- **. How to make SGD Classifier perfomr as well as Logistic Regression using Parfit**
- **. Intuitive Interpretation of Random Forest**
- **. Statoil/C-Core Iceberg Classifier Challenge on Kaggle: a Keras Model for Beginners + EDA**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Thought that was me, that's nice, because when we pup when we posted this on Twitter, you know got a lot of attention and finally, somebody with it was able to say like: oh, you know what this this actually already exists. So Tyler has gone away and you know started reading that book. Something else which is super cool is Jason Carpenter created a whole new library called Parfitt and Parfitt is a parallelized fitting of multiple models for the purpose of selecting hyper parameters and there's a lot. I really like about this he's shown a clear example of how to use it right and, like the API, looks very similar to other grid search based approaches, but it uses the validation techniques that Rachel wrote about and that we learnt about a couple of weeks ago. Of using a good validation set - and you know what he's done here is in his blog post that introduces it, you know, he's he's gone right back and said like what are hyper parameters. Why do we have to train them and he's kind of explained every step and then the the module itself is like it's? It's very polished. You know he's added documentation to it. He's added a nice readme to it and it's kind of interesting when you actually look at the code, you realize you know it's very simple. You know which is it's definitely not a bad thing.

That's a good thing as to is to make things simple, but by kind of writing this little bit of code

and then packaging it up so nicely he's made it really easy for other people to use this technique, which is great, and so one of the things I've been really thrilled to see is then Vinay went along and combined two things from our class. One was to take profit and then the other was to take the kind of accelerated SGD approach to classification. We turn learned about in the last lesson and combine the two to say like okay. Well, let's now use half it to help us find the parameters of a SGD logistic regression. So I think that's really a really great idea. Something else which I thought was terrific is print sexually basically went through and summarized pretty much all the stuff we learnt in the random and random forest interpretation plus, and he went even further than that, as he described each of the different approaches to random forest interpretation. He described how it's done so here, for example, is feature importance, a variable permutation a little picture of each one and then super cool here is the code to implement it from scratch. So I think this is like really nice post. You know describing something that not many people understand and showing you know exactly how it works both with pictures and with code that implements it from scratch. So I think that's really really great.

One of the things I really like here is that for like the tree interpreter, but he actually showed how you can take the tree interpreter output and feed it into the new waterfall chart package that Chris USF student built to show how you can actually visualize the Contributions of the tree interpreter in a waterfall chart so again kind of a nice combination of multiple pieces of technology. We both learned about and and built as a group. I also really thought this kernel. There's been a few interesting kernels share it and I'll share some more next week and diverse wrote this really nice kernel showing this is quite challenging: careful competition on detecting icebergs versus chips and it's a kind of a weird two channel satellite data, which is very hard To visualize - and he actually went through and basically described kind of the formulas for how these, like radar scattering things, actually work and then actually managed to come up with a code that allowed him to recreate. You know the actual 3d icebergs or ships, and I have not seen that done before or like I, you know it's it's quite challenging to know how to visualize his data and then he went on to show how to build a neural net. To try to interpret this so that was pretty fantastic as well. So yeah! Congratulations for all of you! I know for a lot of you.

You know you're posting stuff out there to the rest of the world for the first time you know, and it's kind of intimidating you're used to writing stuff, that you got a hand into a teacher and there any ones who see it - and you know it's kind Of scary, the first time you do it, but then the first time somebody you

Back to the course.

3. 00:09:01

- **Why write a post on your learning experience, for you and for newcomers.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Know that votes your cable kernel or ATS, a clap to your medium post. He suddenly realized. Oh I'm, actually I've written something that people like. That's that's pretty great. So if you haven't tried yourself yet I again invite you to try writing something and if you're not sure you could write a summary of a lesson. You could write a summary of like if there's something

you found hard like. Maybe you found it hard to fire up a gpu-based AWS instance. You eventually figured it out. You know you could write down just describe how you solve that problem or if one of your classmates didn't understand something, and you explained it to them, then you could like write down something saying like oh there's, this concept that some people have trouble understanding. Here's a good way, I think, of explaining it there's all kinds of stuff you could. You could do

4. 00:09:50

- **Using SGD on MNIST for digit recognition**
- **. lesson4-mnist_sgd.ipynb notebook**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, so let's go back to SGD and so we're going back through this logbook, which Rachel put together basically taking us through kind of SGD from scratch for the purpose of digit recognition and actually quite a lot of the stuff we look at today is going to Be closely following part of the computation or linear algebra course, which you can both find the MOOCs on faster, I or at USF it'll, be an elective next year. Alright. So if you find some of this this stuff interesting - and I hope you do - then please consider signing up for the elective or checking out the video online, so we're building neural networks and we're starting with an assumption that we've downloaded the eminence data. We've normalized it by subtracting the main and divided by the standard, deviation. Okay. So the data is it's slightly unusual in that, although they represent images they where they were downloaded as each image was a 784 long rank one tensor, so it's been flattened out, okay and so for the purpose of drawing pictures of it. We had to resize it to 28 by 28, but the

5. 00:11:30

- **Training the simplest Neural Network in PyTorch**
- **(long step-by-step demo, 30 mins approx)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Actual data we've got is not 28 by 28. It says it's, it's 784 long, flattened out. Okay, four basic steps we're gon na take here is to start out with training the world's simplest neural network, basically a logistic regression right, so no hidden layers and we're going to Train it using a library, fastai and we're going to build the network using a Library, plate watch right and then we're going to gradually get rid of all the libraries right. So, first of all well get rid of the N n neural net library and pytorch and write that ourselves then we'll get rid of the fast a I fit function and write that ourselves and then we'll get rid of the pytorch optimizer and write that Ourselves, and so by the end of this notebook, we'll have written all the pieces ourselves. The only thing that will end up relying on is the two key things that pytorch gives us, which is a the ability to write Python code and have it run on the GPU and be the ability to write Python code and have it automatically differentiated. For us, okay, so there are two things: we're not going to attempt to write ourselves because it's boring and pointless, but everything else we'll try and write ourselves on top of those two things. Ok, so our starting point is like not doing anything ourselves. It's basically having it all done for us, and so pytorch has an N n library,

which is where the neural net stuff lives. You can create a multi-layer neural network by using the sequential function and then passing in a list of the layers that you want and we asked for a linear layer, followed by a softmax layer and that defines our logistic regression.

Okay, the input to our linear layer is 28 by 28. As we just discussed, the output is 10 because we want a probability for each of the numbers, not through 9, for each of our images. Okay, CUDA sticks it on the GPU and then fit fits a model. Ok, so we start out with a random set of weights and then fit users gradient descent to make it better. We had to tell the fit function. What criterion to use, in other words, what counts is better and we told it to use negative log likelihood we'll learn about that in the next lesson. What that is exactly we had to tell it what optimizer to use - and we said please use - opt M, not Adam. The details of that we won't cover in this course we're going to use something, build something simpler, called SGD. If you interested in Adam, we just covered that in the dick learning course and what metrics do you want to print out? We decided to print out accuracy, ok, so that was that, and so, if we do that, ok, so after we fit it, we get an accuracy of generally somewhere around 91 92 percent. So what we going to do from here is we're going to gradually we're going to repeat this exact same thing, so we're going to rebuild this model, you know four or five times fitting it, building it and fitting it with less and less libraries. Ok, so the second thing that we did last time was to try to start to define the the module ourselves right. So instead of saying the network is a sequential bunch of these layers. Let's not use that like at all and try and define it ourself from scratch. Okay! So to do that, we have to use our because that's how we build everything in play torch and we have to create a class which inherits from an end module.

So, n, n dot module is a pytorch class that takes our class and turns it into a neural network module which basically means we'll anything that you inherit from an end module like this. You can pretty much insert into a neural network as a layer or you can treat it as a neural network. It's going to get all the stuff that it needs automatically to to work as a part of or a full neural network. Now we'll talk about exactly what that means today in the next lesson right, so we need to construct the object. So that means we need to define the constructor Thunder in it and then importantly, this is a Python thing is, if you inherit from some other object, then you have to create the thing you inherit from first so when you say super dot, dunder init that says Construct the enn module piece of that first right. If you don't do that, then the NN dot module stuff never gets a chance to actually get constructed right. So this is just like a standard Python oo, subclass, constructor, okay, and if any, if that's on unclear to you, then you know this is where you definitely want to just grab a Python intro 200, because this is the standard approach all right so inside our constructor. We want to do the equivalent of an end linea all right, so what n n dot linea is doing? Is it's taking our it's taking our 28 by 28 vector so 768 long vector and we're going to be that's going to be the input to a matrix multiplication.

So we now need to create a something with 768 rows and that's 768 and 10 columns. Ok! So because the input to this is going to be a mini batch of size. Actually, let's move this into a new window 768 by 10, and the input to this is going to be a mini batch of size 64 by 768 right. So we're going to do this matrix product, ok! So when we say in pie chart and in linea it's going to construct this matrix for us right. So since we are not using that we're doing things from scratch, we need to make it ourselves so to make it ourselves, we can say, generate normal random numbers with this dimensionality, which we passed in here, 768 by 10, okay, so that gives us our our randomly Initialized matrix okay, then we want to add on to this. You know we don't just want y equals ax. We want y equals ax plus B, all right, so we need to add on what we call in neural Nets of bias vector. So we create here a bias vector of

length: 10, okay, again randomly initialized, and so now here are our two randomly initialized weight tensors. So that's our constructor okay! Now we need to find for word. Why do we need to define for word? This is a pytorch. Specific thing what's going to happen is this is when you create a module in pytorch, the objects that you get back behaves as if it's a function, you can call it with parentheses, which will do it that a moment, and so you need to somehow define What happens when you call it as if it's a function and the answer is tight, which calls a method called forward? Okay, that's just that! That's the pie that the pytorch kind of approach that they picked right.

So when it calls forward, we need to do our actual calculation of the output of this module or letter. Okay. So here is the thing that actually gets calculated in a logistic regression. So basically we take our input X, which gets passed to forward. That's basically how forward works. It gets past the mini-batch and we matrix multiply it by the layer, one weights which we defined up here and then we add on the layer one bias which we defined up here: okay and actually nowadays, we can define this a little bit more elegantly using the Python three matrix multiplication operator, which is the at sign and when you, when you use that, I think you kind of end up with something that looks closer to what the mathematical notation looked like, and so I find that nicer. Okay, all right! So that's that's our linear layer in our logistic regression. You know a zero hidden layer, neural net. So then the next thing we do to that is soft. Next, okay, so we get the output of this matrix model play. Okay, who wants to tell me what the dimensionality of my output of this matrix model play is sorry 64 by 10. Thank you Karen. I should mention for those of you that weren't at deep learning class yesterday, we actually looked at a really cool post from Karam who described how to do structured data analysis with neural nets, which has been like super popular and a whole bunch of people who kind Of said that they've read it and found it interesting, so that was really exciting.

So we get this matrix of outputs and we put this through a softmax, and why do we put it through a softmax? We put it through a softmax because in the end we want - probably you know, for every image. We want a probability that is a 0 or a 1 or a 2 or 3 or 4 all right. So we want a bunch of probabilities that add up to 1 and where each of those probabilities is between 0 & amp 1. So a softmax does exactly that for us. So, for example, if we weren't picking out you know, numbers from nought to 10, but instead of picking it out, cat dog play an official building. The output of that matrix multiplied for one particular image might look like that. These are just some random numbers and to turn that into a softmax. I first go a to the power of each of those numbers. I sum up those eight of the power offs and then I take each of those eight of the power of z' and divide it by the sum and that softmax that's the definition of softmax so because it was in the power of it means it's always positive, Because it was divided by the sum, it means that it's always between zero and one, and it also means because it's divided by the sum that they always add up to one. So by applying this softmax activation function. So anytime we have a layer of outputs which we call activations, and then we apply some function. Some nonlinear function to that that map's 1:1 scale at a one scalar like softmax.

Does we call that an activation function? Okay, so the softmax activation function takes our outputs and turns it into something which behaves like a probability right. We don't, strictly speaking, need it. We could still try and train something which, where the output directly is the probabilities right, but by creating using this function, that automatically makes them always behave like probabilities. It means there's less for the network to learn, so it's going to learn better alright. So, generally speaking, whenever we design an architecture, we try to

design it in a way where it's as easy as possible for it to create something of the form that we want. So that's why we use softmax right. So that's the basic steps right. We have our input, which is a bunch of images right, which is here. It gets multiplied by a weight metrics. We actually also add on a bias right to get a output of the linear function. We put it through a nonlinear activation function in this case softmax, and that gives us our probabilities, so there there that all is pytorch also tends to use the log of softmax for reasons that don't particularly need father's. Now, it's basically a numerical stability convenience. Okay, so to make this the same as our version up here that you saw log softmax, I'm going to use log here as well. Okay, so we can now instantiate this class. That is create an object of this class. So I have a question back for the probabilities where we were before hmm.

So if we were to have a photo with a cat and a dog together, would that change the way that that works or does it work in the same basic? Yes, that's a great question, so if you had a photo with a cat and a dog together and you wanted it to spit out both cat and dog, this would be a very poor choice. So softmax is specifically the activation function. We use for categorical predictions where we only ever want to predict one of those things right and so part of the reason. Why is that, as you can see, because we're using e to the right e to the slightly bigger numbers creates much bigger numbers? As a result of which we generally have just one or two large and everything else is pretty small right. So if I like recalculate these rounded numbers a few times, you'll see like it tends to be a bunch of zeroes and one or two high numbers right. So it's really designed to try to kind of make it easy to predict like this. One thing is the thing I want if you're doing, multi-label prediction, so I want to just find all the things in this image rather than using softmax. We would instead use sigmoid. That's a sigmoid recall it would cause each of these between to be between 0 & amp 1, but they would no longer add to 1. It's a good question and like a lot of these details about like best practices, are things that we cover in the deep learning course, and we won't cover heaps of them here and the machine learning course we're more interested in the mechanics.

I guess, but we're trying to do them. We've they're, quick, all right so now that we've got that we can instantiate an object of that class and of course we want to copy it over to the GPU, so we can do computations over there again. We need an optimizer we're we talking about what this is shortly, but you'll see here. We've called a function on our class called parameters, but we never defined a method called parameters and the reason that is going to work is because it actually was defined Forest inside an end up module and so an end up module actually automatically go through the attributes. We've created and finds anything that basically we we said this is a parameter. So the way you say something is a parameter. Is you wrap it in an end off parameter? So this is just the way that you tell pytorch. This is something that I want to optimize. Ok, so when we created the weight matrix, we just wrapped it with an end up parameter, it's exactly the same as a regular 5 torch variable which we'll learn about shortly. It's just a little flag to say: hey, you should you should optimize this, and so, when you call net to parameters on our net to object, we created it goes through everything that we created in the constructor checks to see if any of them are of type Parameter and if so, it sets all of those being things that we to train with the optimizer and we'll be implementing the optimizer from scratch later. Okay, so having done that, we can fit and we should get basically the same answers before 91 ish.

So that looks good all right. So what if we actually built here well, what we've actually built, as I said, is something that can behave like a regular function all right. So I want to show you how we can actually call this as a function so to be able to call it as a function. We need to be able to pass data to it to be able to pass data to it. I'm going to need to grab a mini batch of

analyst images. Okay, so we used for convenience the image classifier data from arrays method from fastai, and what that does is it creates a pytorch data loader for us a pytorch data. Loader is something that grabs a few images and sticks them into a mini batch that makes them available, and you can basically say give me another mini batch pick me. Another mini batch, give me another mini batch and so in Python we call these things. Generators generators are things where you can basically say. I want another. I want another. I want another right there's. This kind of very close connection between iterators and generators are not going to worry about the difference between them right now, but you'll see basically to turn to actually get hold of something which we can say. Please give me another of in order to grab something that we can. We can use to generate mini batches. We have to take our data loader, and so you can ask for the training data loader from our model. Data object.

You'll see there's a bunch of different data loader, as you can ask, for you can ask for the test data loader, the Train date, loader, the validation, loader or wintered images, data, loader and so forth. So we're going to grab the training data loader that was created for us. This is a pice and plate, or data loader well slightly optimized by us, but same idea, and you can then say this is a standard Python thing we can say turn that into an iterator turn that into something where we can grab another one at a time From and so once you've done that we've now got something that we can iterate through. You can use the standard Python next function to grab one more thing from that generator: okay, so that's returning and the X's from a mini-batch and the Y's found our mini batch. The other way that you can use generators and iterators in python is with a for loop. I could also said like, for you know, X, mini batch, comma Y mini batch in data loader and then like do something right. So when you do that, it's actually behind the scenes - it's basically syntactic sugar for calling next lots of times. Okay. So this is all standard Python stuff, so that returns a tensor of size 64 by 784, as we would expect right, the the FASTA I library we used defaults to a mini batch size of 64. That's why it's that long! These are all of the background. 0 pixels, but they're not actually zero.

In this case, why aren't they zero yeah they're normalized exactly right, so we subtracted the mean divided by the standard, deviation right, so there there it is so now what we want to do is we want to pass that into our our logistic regression? So what we might do is we'll go variable X, M B equals variable. Okay, I can take my X mini-batch. I can move it onto the GPU because remember my net to object is on the GPU, so our data, for it also has to be on the GPU and then the second thing I do is: I have to wrap it in variable. So what is variable? Do this is how we get for free automatic differentiation. pytorch can automatically differentiate. You know pretty much anything right, any tensor, but to do so takes memory and time. So it's not going to always keep track like to do what have any differentiation. It has to keep track of exactly how something was calculated. We added these things together, we multiplied it by that. We then took the sign blah blah blah right. You have to know all of the steps, because then to do the automatic differentiation it has to take the derivative of each step using the chain rule, multiply them all together right. So that's slow and memory intensive. So we have to opt in to saying like okay. This particular thing we're going to be taking the derivative of later so please keep track of all of those operations for us, and so the way we opt-in is by wrapping a tensor in a variable right.

So that's how we do it and you'll see that it looks almost exactly like a tensor, but it now says variable containing this tensor right. So in pytorch, a variable has exactly identical api to a tensor or actually more specifically a superset with the api of a tensor anything we can do to a tensor. We can do to a variable, but it's going to keep track of exactly what we did. So we can later on, take the derivative okay, so we can now pass that into our net to object. Remember I

said you can treat this as if it's a function right so notice we're not calling dot forward, we're just treating it as a function and then remember. We took the log so to undo that I'm taking the X and that will give me my probabilities. Okay, so there's my probabilities and it's got return something of size 64 by 10. So for each image in the mini batch, we've got ten probabilities and you'll see. Most probabilities are pretty close to zero right, and a few of them are quite a bit bigger, which is exactly what we do. We're hope right is that it's like okay, it's not a zero, it's not a one. It's not a two! It is a three. It's not a four, it's not a five and so forth. So maybe this would be a bit easier to read if we just grabbed like the first three of them: okay, just like 10 to the negative, the neg two five, five, four okay and then suddenly here's one which is ten to Nick one right.

So you can kind of see what it's trying to I was trying to do here I mean we could call like net two dot forward and it will do exactly the same thing right, but that's not how all of the pytorch mechanics actually work. It's actually. They actually call it as if it's a function right, and so this is actually a really important idea like because it means that when we define our own architectures or whatever anywhere that you would put in a function you could put in a layer anyway, you put In a layer you can put in a neural net anyway, put it on your neck. You can put in a function because, as far as pytorch is concerned, they're all just things that it's going to call just like as if they're functions so they're all like interchangeable - and this is really important, because that's how we create really good neural nets is By mixing and matching lots of pieces and putting them all together right, let me give an example. Here is my logistic regression which got 91 and a bit percent accuracy. I'm now going to turn it into a neural network with one hidden layer, all right and the way I'm going to do. That is I'm going to create my more layer, I'm going to change this, so it's spits out a hundred rather than ten, which means this one input is going to be a hundred rather than ten.

Now this as it is, can't possibly make things any better at all, yet why is this definitely not going to be better than what I had before yeah? Can somebody pass the yeah bishop, your combination of two linear layers, which is just the same as exactly right? So we've got two linear layers, which is just a linear layer right so to make things interesting, I'm going to replace all of the negatives from the first layer with zeros, because that's a nonlinear transformation and so that nonlinear transformation is called a rectified linear unit. Okay, so n n dot sequential simply is going to call each of these layers in turn for each mini batch right. So dual linear layer replace all of the negatives with zero, do another linear layer and do it softbank's. This is now a neural network with one hidden layer, and so let's try trading that instead, yeah accuracies now been up to 96 %, okay, so the this is. The idea is that the basic techniques we're learning in this lesson like become powerful at the point where you start stacking them together. Okay, can somebody pass the green box there and then yes, Daniel, no reason it was like easier to type an extra zero. Like this question of like how many activations should I have a neural network, layer is kind of part of the the scale of a deep learning practitioner we covered in the deep learning course and not in this class.

When adding that additional, I guess transformation additional layer. Additional this one here is called a nonlinear layer or an activation activation function. Direct activation function. Does it matter that like if you would have done for like to soft Max's, or is that something you cannot do like yo? You can absolutely use the softmax there, but it it's probably not gon na give you what you want and the reason why is that a soft max tends to push most of its activations to zero and an activation just be clear, like I've had a lot of Questions in deep learning course about like what's an activation. An activation is the value that is calculated in a layer right. So this is an activation right. It's not a weight. A weight is not an activation. It's the

value that you calculate from a layer, so soft max will tend to make most of its activations pretty close to zero, and that's the opposite of what you want. You generally want your activations to be kind of as rich and diverse and and used as possible, so nothing to stop you doing it, but it probably won't work very well. Basically, pretty much all of your layers will be followed by non by nonlinear activation functions. That will nearly always be value, except for the last layer. It's going to or three layers deep. Do you want to switch up these activation layers? That's a great question. So, if I wanted to go deeper, I would just do that.

Okay, that's an outer hidden layer Network, so I think I'd heard you said that there are a couple of different activation functions like that rectified linear units. What are some examples, and why would you use each yeah great question so basically like, as you add, like more linear layers, you kind of got your input comes in and you put it through a linear layer and then a nonlinear layer, linear layer, nonlinear layer learning A linear layer and then the final nonlinear layer, the final nonlinear layer, as we've discussed, you know if it's a multi category classification, but you only ever pick one of them. You would use softmax if it's a binary classification or a multi-label classification, where you're predicting multiple things. You would use sigmoid. If it's a regression, you would often have nothing at all right, although we learnt in last night's DL course where sometimes you can use sigmoid there as well so they're, basically the options main options for the final layer for the hidden layers. You pretty much always use value right, but there is a another another one. You can pick, which is kind of interesting, which is called leaky value, and it looks like this and basically, if it's above zero, it's y equals x and if it's below zero. It's like y equals 0.1 X, okay, so that's very similar to value, but it's you know, rather than being equal to 0 under X. It's it's like something close to that so they're, the main to rally and lake here Lu. There are various others, but they're kind of like things that just look very close to that. So, for example, there's something called Lu, which is quite popular but, like you know the details, don't matter too much honestly like that they're like aou, is something that looks like this, but it's slightly more curvy in the middle and it's kind of like it's not generally, Something that you so much pick based on the data set, it's more like over time.

We just find better activation functions so two or three years ago, everybody is value. You know a year ago, pretty much everybody used Lake Erie Lu today I guess probably most people are starting to move towards ALU, but honestly that the choice of activation function doesn't matter terribly much actually, and you know, people have actually showed that you can use like A pretty arbitrary nonlinear activation functions like even a sine wave, and it still works okay. So, although what we're going to do today is showing how to create this network with no hidden layers to turn it into that Network, which is 96 % ish accurate, is it will be trivial right and in fact it's something. You should probably try and do during the week right is to create that version. Okay, so now that we've got something where we can take our network parsing our variable and get back some predictions, that's basically all that happened when we called fish. So we're going to see how how that that approach can be used to create this to cast a gradient descent. One thing to note is that the to turn the predicted probabilities into a predicted like which digit is it we would need to use AG max. Unfortunately, pytorch doesn't call it ad max instead, pytorch just calls it max and max returns two things: it returns the actual max across this axis. So this is across the columns right and the second thing it returns is the index of that maximum right.

So so the equivalent of arc max is to call max and then get the first indexed thing. Okay, so there's our predictions right. If this was in numpy, we would instead use MP. Dot admits. Okay, all right. So here are the predictions from our hand, created logistic regression and, in

this case, looks like we've got all, but one correct. So the next thing we're going to try and get rid of in terms of using libraries is for try to avoid using the matrix multiplication operator and instead we're going to try and write that by hand.

6. 00:46:55

- **Intro to Broadcasting: "The MOST important programming concept in this course and in Machine Learning"**
- **. Performance comparison between C and Python**
- **. SIMD: "Single Instruction Multiple Data"**
- **. Multiple processors/cores and CUDA**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So this next part we're going to learn about something which kind of seems it kind of it's. Gon na seem like a a minor little kind of programming idea, but actually it's going to turn out that, at least in my opinion, it's the most important programming concept that will teach in this course and it's possibly the most important programming kind of concept. In all of all the things you need to build machine learning, algorithms and it's the idea of broadcasting and the idea I will show by example, if we create an array of 10 6 NIC 4 and an array of 2 8 7 and then add the two Together, it adds each of the components of those two arrays in turn. We call that element wise. So in other words, we didn't have to write a loop right back in the old days. We would have to have looped through each one and added them and then concatenate them together. We don't have to do that today. It happens for us automatically. So in lump a we automatically get element wise operations. We can do the same thing with pytorch, so in first day I would just add a little capital T to turn something into a pipe watch. Tensor all right and if we add those together exactly the same thing, all right so element wise operations are pretty standard in these kinds of libraries. It's interesting not just because we don't have to write the for loop right, but it's actually much more interesting because of the performance things that are happening here.

The first is, if we were doing a for loop right, if we were doing a four loop that would happen in Python right. Even when you use play torch, it still does the for loop in Python. It has no way of like optimizing a for loop, and so a for loop in Python is something like 10,000 times slower than in C. So that's your first problem, like I remember, is like 1,000 or 10,000. The second problem, then, is that you don't just want it to be optimized in C, but you want C to take advantage of the thing that your all of your CPUs do to something called Cindy single instruction, multiple data, which is yours, your CPU, is capable of Taking eight things at a time right in a vector and adding them up to another vector with eight things in in a single CPU instruction right. So if you can take advantage of Sim D you're immediately eight times faster, it depends on how big the data type is. It might be, four might be eight. The other thing that you've got in your computer is you've, got multiple processes, multiple cores, so you've probably got like. If this is inside happening on one side, one core you've probably got about four of those okay. So if you're using Cindy your eight times faster, if you can use multiple cores than your 32 times faster and then, if you're doing that in C, you might be something like 32 times per thousand times faster right. And so the nice thing is that when we do that, it's taking advantage of all of these things - okay, better still, if you do it in pytorch and your data was created with CUDA to stick it on the GPU, then your GPU can do about 10,000. Things at a time all right so that'll be another hundred times faster than C all right, so this is critical to

getting good performance. Is you have to learn how to write Lewis code by taking advantage of these element-wise operations and like it's not it's a lot more than just plus.

I can also use less then right and that's going to return 0, 1 1 or if we go back to numpy false true true, and so you can kind of use this to do all kinds of things without looping. So, for example, I could now multiply that by a and here are all of the values of a as long as they're less than B or we could take the mean this is the percentage of values in AE that are less than B, all right. So, like there's a lot of stuff, you can do with this simple idea, but to take it further right to take it further than just this element: wise operation, we're going to

7. 00:52:10

- **Broadcasting in details**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Have to go the next step to something called broadcasting, so, let's take a five-minute break, come back at 2:17 and we'll talk about broadcasting so broadcasting. This is the definition from the numpy documentation of broadcasting and I'm going to come back to it in a moment rather than reading it now. But let's start by looking an example of broadcasting. So a is a array with one dimension, also known as a Rank: 1 tensor, also known as a vector we can say a greater than 0. So here we have a Rank 1 tensor right and a rank 0 tensor right. A rank. 0 tensor is also called a scalar. The rank 1 tensor is also called a vector, and we've got an operation between the two all right. Now, you've probably done it a thousand times without even noticing that's kind of weird right that you've got these things of different ranks and different sizes. So what is it actually doing right? But what it's actually doing is it's, taking that scaler and copying it here-here-here right and then it's actually going element-wise tan is greater than zero. Six is greater than zero. Minus pore is greater than zero. You have been giving us back the three answers right and that's called broadcasting broadcasting means copying one or more axes of my tensor to allow it to be the same shape as the other tensor.

It doesn't really copy it, though, what it actually does is it stores this kind of internal indicator that says pretend that this is a vector of three zeros, but it actually just like, rather than kind of going to the next row, we're going to the next scaler. It goes back to where it came from if you're interested in learning about this. Specifically it's they set the stride on that axis to be zero. That's a minor advanced concept for those who procures, so we could do a plus one right. It's going to broadcast the scalar 1 to be 1 1, 1 and then do element wise addition. If we could do the same with a matrix right, here's our matrix 2 times the matrix is going to broadcast to to be to to to to to to to to to and then do element wise multiplication right. So that's our kind of most simple version of broadcasting, so here's a slightly more complex version of broadcasting. Here's an array called C right. So this is a rank. One tensor and here's our matrix M from before rank two tensor. We can add M plus C alright. So, what's going on here, one two, three: four: five: six: seven, eight nine, that's M all right and then C: 10. 20. 30. You can see that what it's done is to add that to each row right 11, 22, 33, 14, 25, 36, and so we can kind of figure. It seems to have done. The same kind of idea is broadcasting a scaler. It's like made copies of it and then it treats those as if it's a rank two matrix, and now we can do element wise addition that make sense right now.

That's yes can. Can you pass that Devin over there? Thank you so it's like by looking at this

example. It like copies it done, making new rows. So how would we want to do it if we wanted to get new columns, I'm so glad you asked so. Instead we would do this 10. 20. 30. All right and then copy that 10, 20, 30, 10, 20, 30 and now treat that as our matrix, so to get numpy to do that, we need to not pass in a vector but to pass in a matrix with one column, a rank, two tensor right. So basically, it turns out that numpy is going to think of a rank 1 tensor for these purposes, as if it was a rank, two tensor, which represents a row right, so in other words, that it is 1 by 3. All right, so we want to create a tensor which is 3 by 1 there's a couple of ways to do that: one is to use NP expand, imps and if you then pass in this argument, it says please insert a length 1 axis here, please so in Our case, we want to turn it into a 3 by 1, so if we said expanding c comma 1, okay, so if we say expanding C comma one, it changes the shape to three comma one. So if we look at what that looks like that looks like a column okay, so if we now go that plus M, you can see it's doing exactly what we hoped it would do, alright, which is to add ten, twenty thirty to the column. Ten, twenty thirty to the column, ten, twenty thirty to the column.

Okay. Now because the location of a unit axis turns out to be so important, it's really helpful to kind of experiment with creating these extra unit axes and know how to do it easily and MP. Dot, expand ins isn't, in my opinion, the easiest way to do this. The easiest way the easiest way is to index into the tensor with a special index. None and what none does is. It creates a new axis in that location of length, one right. So this is going to add a new axis at the start of length, one. This is going to add a new axis at the end at length one or why not they're both right. So if you think about it like a tensor which has like three things in it could be of any rank, you like right, you can just add, you know, taxis all over the place, and so that way we can kind of decide how we want our broadcasting To work so there's a pretty convenient thing in numpy called broadcast, and what that does is it takes our vector and broadcasts it to that shape and shows us what that would look like right. So if you ever like unsure of what's going on in some broadcasting operation, you can save broadcast too, and so, for example, here we could say like rather than three comma three, we could say MJ right and see exactly what's happening, gon na happen, and so that's. What's gon na happen before we add it to em right so if we said turn it into a column, that's what that looks like makes sense, so that's kind of like the intuitive definition of broadcasting, and so now, hopefully we can go back to that numpy documentation And understand what it means right broadcasting describes how numpy is going to treat arrays of different shapes when we do some operation right.

The smaller array is broadcast across the larger array. By smaller array, they mean lower rank tensor, basically, our broadcast across the light. The higher rank tensor, so they have compatible shapes it vector, eise's array operation, so vectorizing generally means like using sim D and stuff like that, so that multiple things happen. At the same time, all the looping occurs in C, but it doesn't actually make needless copies of theta. It kind of just acts as if it had okay. So there's our definition. Now, in deep learning, you very often deal with tensors of rank four or more, and you very often combine them with tensors of rank one or two and trying to just rely on intuition to do that correctly as nearly impossible. So you really need to know the rules. So here are the rules. Okay, here's my shop, here's C dot shape. So the rule are that we're going to compare the shapes of our two tensors element-wise, we're going to look at one at a time and we're going to start at the end. All right so look at the trailing dimensions and then go towards the front. Okay and so two dimensions are going to be compatible when one of these two things is true right. So let's check right, we've got our our M & amp C. Compatible m is 3. 3 c is 3 right, so we're going to start at the end, trailing dimensions first and check. Are they compatible they're compatible if the dimensions are equal? Okay, so these ones are equal, so they are compatible right. All right.

Let's go to the next one. Uh-Oh we're missing all right C is missing something. So what happens if something is missing, as we insert a 1 okay, that's the rule, all right, and so let's now check are these compatible. One of them is one yes, they're compatible. Okay, so now you can see why it is that numpy treats the one dimensional array as if it is a rank, two tensor which is representing a row. It's because we're basically inserting a one at the front. Okay, so that's the rule, so, for example, this is something that you very commonly have to do, which is you start with like an image there, like 256 pixels by 256 pixels by 3 channels, and you want to subtract the mean of each channel right. So you've got 256 by 256 by 3 and you want to subtract something of length 3 right. So yeah. You can do that absolutely because 3 & amp, 3 are compatible because they're the same right, 256 and empty is compatible. It's going to insert a 1 256 and empty is compatible. It's going to insert of 1 okay, so you're going to end up with this is going to be broadcast over all of this access and then that whole thing will be broadcast over this access and so we'll end up with a 256 by 256 by 3, effective Tensor here right so, interestingly, like very few people in the data science or machine learning, communities, understand broadcasting and the vast majority of the time.

For example, when I see people doing pre-processing for computer vision like subtracting the mean they always write loose over the channels right - and I kind of think like it's it - it's like so handy to not have to do that, and it's often so much faster to not Have to do that so if you get good at broadcasting, you'll have this like super useful skill that very very few people have and and

8. 01:05:50

- **Broadcasting goes back to the days of APL (1950's) and Jsoftware**
- **. More on Broadcasting**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Like it's, it's it's an ancient skill. You know it goes. It goes all the way back to the days of APL, so APL was from the late. 50S stands for a programming language and Kenneth Iverson wrote this paper called notation as a tool for thought in which he proposed a new math notation, and he proposed that if we use this new math notation, it gives us new tools for thought and allows us to Think things we couldn't before and one of his ideas was broadcasting not as a computer programming tool but as a piece of math notation, and so he ended up implementing this notation as a tool for thought. As a programming, language called APL and his son has gone on to further develop that into a piece of software called J, which is basically what you get when you put sixty years of very smart people working on this idea, and with this programming language, you can Express very complex mathematical ideas, often just where the line of code or two, and so I mean it's great, that we have J, but it's even greater that these ideas have found their ways into the languages we all use like in Python. The numpy and pytorch libraries all right - these are not just little kind of niche ideas is like fundamental ways to think about math and to do programming. Let me give an example of like this kind of notation as a tool for thought. Let's, let's look here: we've got C right here: we've got C, none right notice. This is now up to square brackets right, so this is kind of like a one row vector tensor.

Here it is a little column. So what is round ones? Okay? What's that gon na do ever think about it, anybody want to have a go, can't even talk through your thinking. Okay, can we pass

the check over there? Thank you. Yes, absolutely so. Take us through your thinking, how's, that going to work so the diagonally lumens can be directly visualized from the squares then cross 1020 clothes, 20 and 30 plus 30. And if you multiply the first row for this column, you get the first row of the matrix mm-hmm. So, finally, we will get our 3 cross 3 matrix, yeah and so to think of this in terms of like those broadcasting rules, we're basically taking this column right, which is a rank 3 comma 1 right and this kind of roll sorry have dimension 3, comma 1 And this row, which is of dimension 1, comma, 3 right and so to make these compatible with our broadcasting rules right. This one here has to be duplicated 3 times because it needs to match this. Okay and now this one's going to have to be duplicated three times to match this okay, and so now I've got two matrices to do an element-wise product of, and so, as you say there is that outer product right now. The interesting thing here is that suddenly now that this is not a special mathematical case, but just a specific version of the general idea of broadcasting we can do like and out a plus or we can do an order greater than right or whatever right.

So it's suddenly we've kind of got this this this concept, that we can use to build new ideas, and then we can start to experiment with those new ideas, and so you know, interestingly, numpy actually uses this. Sometimes, for example, if you want to create a grid, this is how numpy does it all right? Actually, this is kind of the sorry. Let me show you this way. If you want to create a grid, this is how numpy does it. It actually returns zero. One two three four and zero one, two three four one is a column. One is a row, so we could say like okay, that's X, grid, comma Y grid, and now you could do something like row. I mean we could obviously go like that right and so suddenly we've expanded that out into a grid right and so yeah. It's kind of interesting how like some of these, like simple little concepts, kind of get built on and built on and built on. So if you lose something like APL or J, it's this whole environment of layers and layers and layers of this. We don't have such a deep environment in numpy, but you know you can certainly see these ideas of like broadcasting coming through in in simple things like how do we create a grid in non-pay, so yeah? So that's that's.

9. 01:12:30

- **Matrix Multiplication -and not-.**
- **. Writing our own training loop.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Broadcasting - and so what we can do with this now is use this to implement matrix multiplication ourselves. Ok, now, why would we want to do that? Well, obviously, we don't write matrix. Multiplication has already been handled perfectly nicely for us by our libraries, but very often you'll, find in all kinds of areas in in machine learning and particularly in deep learning that there'll be particular types of linear function that you want to do that aren't quite done for You right so, for example, there's like whole areas called like tensor regression and tensor decomposition, which are really being developed a lot at the moment and they're kind of talking about, like how do we take like higher rank, tensors and kind of turn them into combinations of Rows columns and faces - and it turns out that when you can kind of do this, you can basically like deal with really high dimensional data structures with not much memory and not with not much computation time. For example, there's a really terrific library called tensile e, which does a whole lot of this kind of stuff for you. So it's a really really important area. It covers like all of deep learning lots of modern machine learning in general,

and so even though you're not going to like define matrix multiplication, you're very likely to wanted to find some other slightly different tensor product. You know. So it's really useful to kind of understand how to do that.

So let's go back and look at our matrix and our our 2d array in 1d array rank two tensor rank 1 tensor and remember. We can do a matrix multiplication using the @ sign or the old way and PMML okay, and so what that's actually doing when we do that is we're basically saying ok, 1 times 10 plus 2 times 20 plus 3 times 30 is 140 right, and so we Do that for each row and we can go through and do the same thing for the next one and for the next one to get our result right. You could do that in torch as well. We could make this a little shorter, okay, same thing, okay, but that is not matrix multiplication. What's that, okay element, wise specifically, we've got a matrix and a vector so broadcasting, okay, good, so we've got this is element wise with broadcasting but notice the numbers it's created. 10. 40. 90. Are the exact three numbers that I needed to calculate when I did that first piece of my matrix multiplication? So in other words, if we sum this over the columns, which is axis equals 1, we get our matrix vector product okay. So we can kind of do this stuff without special help from our library. So now, let's expand this out to a matrix matrix product. So a matrix, matrix product looks like this. This is this great site called matrix multiplication XYZ and it shows us this is what happens when we multiply two matrices.

Okay, that's what matrix multiplication is operationally speaking, so, in other words, what we just did there was we first of all took the first column with the first row to get this one, and then we took the second column with the first row to get that one Right so we're basically doing the thing we just did the matrix vector product we're just doing it twice right once with this column and once with this column, and then we concatenate the two together okay, so we can now go ahead and do that, like so M Times the first column dot some M times the second column, that's some, and so there are the two columns of our matrix multiplication. Okay, so I didn't want to like make our code too messy, so I'm not going to actually like use that, but like we have it there now, if we want to, we don't need to use torch or numpy matrix multiplication anymore. We've got, we've got our own! That we can use using nothing but element wise operations, Broadcasting and some okay. So this is our logistic regression from scratch. Class again I just copied it here: here's where we instantiate the object copy to the GPU. We create an optimizer which we'll learn about in a moment and we call fit okay. So the goal is to now repeat this without needing to call fit so to do that. We're going to need a loop which grabs a mini batch of data at a time and with each mini batch of data.

We need to pass it to the optimizer and say: please try to come up with a slightly better set of predictions for this mini batch right. So, as we learnt in order to grab a mini batch of the training set at a time, we have to ask the model data object for the training data loader. We have to wrap it an iterator to create an iterator or a generator, and so that gives us our our data. Loader. Okay, so pytorch calls this a data loader. We actually wrote our own class today as a loader, but it's it's all. It's basically the same idea, and so the next thing we do is we grab the X and the y tensor the next one from our data. Loader, okay wrap it in a variable to say: I need to be able to take the derivative of the calculations using this, because if I can't take the derivative, then I can't get the gradients and I can't update the weights all right and I need to put It on the GPU, because my module is on the GPU, and so we can now take that variable and pass it to the objects that we instantiated our logistic regression remember now module we can use it as if it's a function, because that's how high-touch works and That gives us a set of predictions, as we saw on scene before okay. So now we can check the loss and the loss, we're defined as being a negative log likelihood, loss object, okay and we're going to learn

about how that's calculated in the next lesson and for now think of it.

Just like a root mean squared error, but for classification problems, so we can call that also just like a function, so you can kind of see this. It's very general idea in pytorch that you know kind of treat everything ideally like it's a function. So in this case we have a loss, negative log likelihood loss object. We could treat it like a function, we pass in our predictions and we pass in our actuals all right again. The actuals need to be turned into a variable and put on the GPU, because the loss is specifically the thing that we actually want to take the derivative of right, so that gives us our loss and there it is that's our loss to point four three. Okay, so it's a variable and because it's a variable, it knows how it was calculated all right. It knows it was calculated with this loss function. It knows that the predictions were calculated with this network. It knows that this network consisted of these operations, and so we can get the gradient automatically and so to get the gradient recall, I'll drop backward. Remember, L is the thing that contains our loss right, so L drop backward is, is something which is added to anything. That's a variable, you even called drop backward, and that says please calculate the gradients, okay and so that calculates the gradients and stores them inside that that the basically for each of the weights that was used. It used each of the parameters that was used to calculate that it's now stored a dot grad.

Well, we'll see it later. It's basically stored the gradient right, so we can then call optimizer dot step and we're going to do this bit step manually shortly and that's the bit that says: please make the weights a little bit better right, and so what optimizer dot step is doing. Is it saying like okay, if you had like a really simple function like this right, then what the optimizer does is it says: okay, let's pick a random starting point right and let's calculate the value of the loss right. So here's our parameter. Here's our loss right! Let's take the derivative right, the derivative tells us which way is down, so it tells us. We need to go that direction. Okay and we take a small step, and then we take the derivative again and we take a small step derivative again. Take a small step drove it again, take a small step until eventually we're taking such small steps that we stop. Okay. So that's what gradient descent does? Okay, how big a step is a small step. Well, we basically take the derivative here. So let's say derivative, there is like eight right and we multiply it by a small number like say 0.01, and that tells us what step size to take. This small number here is called the learning rate and it's the most important hyper parameter to set right. If you pick two smaller learning rate, then your steps down are going to be like tiny and it's going to take you forever.

Okay, too big a learning rate and your jump too far right and then you'll jump too far and your diverge rather than converge. Okay, we're not going to talk about how to pick a learning rate in this class, but in the deep learning class we actually show you a specific technique that very reliably picks a very good learning rate. So that's basically what's happening right, so we calculate the derivatives and we call the optimizer that does a step, in other words, update the weights based on the gradients and the learning rate. We should hopefully find that, after doing that, we have a better loss than we did before, so I just really ran this and got a loss here of 4.16 and after one step it's now 4.0. 3. Okay, so it worked the way we hoped it word based on this mini batch. It updated all of the weights in our network to be a little better than they were, as a result of which our loss went down. Okay, so let's turn that into a training loop, all right, we're going to go through a hundred steps grab one more mini batch of data from the data loader calculate our predictions from our network calculate our loss from the predictions and the actuals. Every 10 goes we'll print out the accuracy just take the mean of the whether they're, equal or not, 1 pi taut, specific thing you have to 0 the gradients. Basically, you can have networks where, like you've, got lots of different loss functions that

you might want to add all of the gradients together right.

So you have to tell play torch like when to set the gradients back to zero right, so this just says: set all the gradients to zero calculate the gradients, let's quote backward and then take one step of the optimizer, so update the weights using the gradients and The learning rate - and so once we run it, you can see, the loss goes down and the accuracy goes up. Okay, so that's the basic approach, and so next lesson we'll see what that does alright. Well, we're looking in detail we're not going to look inside here, as I say, we're going to basically take the calculation of the derivative says, that's a given right, but basically what's happening there in any kind of deep network. You have kind of like a function. That's, like you know, a linear function, and then you pass the output of that into another function. That might be like a rally and you pass the output of that into another function that might be another linear net Lenny Olea. You pass that into another function. That might be another value and so forth right. So at these, these deep networks are just functions of functions of functions, so you could write them. Mathematically like that right and so all backprop does is. It says, let's just simplify this down to the two version, as we can say: okay, u equals f of X right and so therefore the derivative of G of f of X is we can calculate with the chain rule as being G. You f dash X, right, and so you can see we can do the same thing for the functions of the functions of the functions, and so when you apply a function to a function of a function, you can take the derivative just by taking the product of The derivatives of each of those layers, okay and in neural networks.

We call this back propagation okay. So when you hear back propagation, it just means use the chain rule to calculate the derivatives, and so when you see a neural network to find like here right like if it's defined sequentially literally all this means is apply. This function to the input apply this function to that apply. This function to that apply this function to that right, so this is just defining a composition of a function, to a function, to a function, to a function, okay and so yeah. So, although we're not going to bother with calculating the gradients ourselves, you can now see why it can do it right as long as it has internally. You know a it knows like: what's the what's the derivative of ^, what's the derivative of sine, what's the derivative of + and so forth, then our Python code in here is just combining those things together, so it just needs to know how to compose them together With the chain rule and where it goes, okay, okay, so I think we can leave it there for now and yeah and in the next class, we'll go and we'll see how to write our own optimizer and then we'll have solved em mist from scratch. Ourselves see you then

Lesson 10

Outline

- Rewriting the 1-layer NN from scratch
- Rewrite LinearLayer
- Rewrite Softmax
- Understanding numpy and torch matrix operations
- Understanding Broadcasting rules
- Rewriting matrix mult from scratch
- Start looking at the fit function

Video Timelines and Transcript

1. 00:00:01

- **Fast.ai is now available on PIP !**
- **And more USF students publications: class-wise Processing in NLP, Class-wise Regex Functions**
- **. Porto Seguro's Safe Driver Prediction (Kaggle): 1st place solution with zero feature engineering !**
- **Dealing with semi-supervised-learning (ie. labeled and unlabeled data)**
- **Data augmentation to create new data examples by creating slightly different versions of data you already have.**
- **In this case, he used Data Augmentation by creating new rows with 15% randomly selected data.**
- **Also used "auto-encoder": the independant variable is the same as the dependant variable, as in "try to predict your input" !**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Well, welcome back to machine learning, one of the most exciting things this week, almost certainly the most exciting. The thing this week is that fastai is now on Pitt, so you can pip install fastai, and so thank you to Prince and perk to creme for making that happen to USF students who had never published a hit package before - and this is one of The harder ones to publish because it's got a lot of dependencies, so it's you know, probably still easiest just to do the Condor end of update thing, but a couple of places that it would be handy instead to pip install fastai would be well. Obviously, if you're working outside of the the repo in the notebooks, then this gives you access to fastai everywhere also, I believe they submitted a pull request to CAG or to try and get it added to the Capitol kernels, so hopefully you'll be able to use It on Kapil kernels soon and yeah. You can use it at your work or whatever else. So that's that's exciting. I mean I I'm not gon na say it's like officially released. Yet you know it's still very early, obviously, and we're still you're helping, add documentation and all that kind of stuff. But it's great that that's now there a couple of cool kernels from USF students

this week thought of highlight two that were both from the text: normalization competition, which was about trying to take text which was written out. You know, wrote a standard English text.

They also had one per Russian and you're trying to kind of identify things that could be like first. Second, third and say like that's a cardinal number, or this is a phone number or whatever, and I did a quick little bit of searching and I saw that there had been some attempts in academia to use deep learning for this. But they hadn't managed to make much progress and actually noticed so up here is Colonel here, which gets 0.99 two on the leader board, which i think is like top is yeah it's kind of entirely heuristic and it's a great example of kind of feature engineering. That's in this case, the whole thing is basically entirely feature engineering, so it's basically looking through and using most of regular expressions to figure out for each token. What is it you know, and I think she's done a great job here, kind of laying it all out. Clearly, as to what all the different pieces are and how they all fit together - and she mentioned that she's - maybe hoping to turn this into a library which I think would be great right - you know you could use this to grab a piece of text and pull Out what are all the pieces in it? It's the kind of thing that the the natural language can, like natural language processing community hopes to be able to do without, like lots of hand, written code like this, but for now this is I'll, be interesting to see. Like what the winners turn out to have done, but I haven't seen machine learning being used really to do this, particularly well, perhaps the best approaches or ones which combine this kind of feature engineering along with some machine learning.

But I think this is a great example of effective feature engineering, and this is a another USF student who has done much. The same thing got a similar kind of score, but used used her own different set of rules. Again this is gets you. It would get you a good leader board position with these as well. So I thought that was interesting to see examples of some of our students entering a competition and getting kind of top 20 ish results by you know basically just handwritten heuristics, and this is where, for example, computer vision was six years ago. Still, basically, all the best approaches was a whole lot of like carefully handwritten heuristics, often combined with some simple machine learning, and so I think, over time you know the field is kind of definitely trying to move towards automating much more of this, and actually, interestingly, very Interestingly, in the safe driver, diction competition was just finished. One of the Netflix Prize winners won this competition and he invented a new algorithm for dealing with structured data which basically doesn't require any feature engineering at all. So he came first place using nothing but five. Deep learning models and one gradient, boosting machine and his his basic approach was very similar to what we've been learning in this class so far.

And what we'll be learning also tomorrow, which is using fully connected neural network somewhere and one hot encoding and specifically embedding which we'll learn about? But he had a very clever technique, which was. There was a lot of data in this competition which was unlabeled. So, in other words, where they didn't know whether that driver would go under claim or not or or whatever so unlabeled data. So when you've got some labeled in some unlabeled data, we call that semi-supervised learning and in real life, most learning is semi-supervised. Learning like in real life, normally you have some things that are labeled and some things that are unlabeled, so this is kind of the most practically useful kind of learning and then structured data is it's the most common kind of data that companies deal with day to Day so the fact that this competition was a semi-supervised structured data, competition made it incredibly practically useful, and so what his technique for winning this was was to through data augmentation, which those of you doing. The deep learning course have learned about,

which is basically the idea. Like if you had pictures, you would like flip them horizontally or rotate them a bit later. Orientation means creating new data examples which are kind of slightly different versions of ones. You already have and the way he did it was for each row from the data.

He would like at random replaced 15 % of the variables with a different row, so each row now would represent like a mix of like 80 %, 85 % of the original row, the 15 % randomly selected from a different, and so this was a way of Like randomly changing the data a little bit and then he used something called an autoencoder which we will probably won't study into a part too, with a deep learning course. But the basic idea of an autoencoder is your dependent variable is the same as your independent variable. So, in other words, you try to predict your input, which obviously is trivial. If you're allowed to like, like you know the identity transporting, for example, Rivoli predicts the input, but the trick with an autoencoder is to have less activations in at least one of your layers than your input right. So if your input was like a hundred dimensional vector - and you put it through a 100 by 10 matrix, okay, create ten activations and then have to recreate the original 100 long vector from that, then you've basically come. You had to have compressed it effectively, and so it turns out that that kind of neural network you know, is forced to find correlations and features and interesting relationships in the data, even when it's not labeled, so he used that rather than doing any, he didn't do Any hand engineering he just used an autoencoder.

So you know these are some interesting kind of directions that if you keep going with your machine learning studies, you know, particularly if you do a part two with a deep learning course next year, your your lone about - and you can kind of see how feature engineering Is going away - and this was just yeah an hour ago - so this is very recent using to eat, but it's one of this is one of the most important breakthroughs I've seen in a long time. Okay, so we were working

2. 00:08:30

- **Back to a simple Logistic Regression with MNIST summary**
- **'lesson4-mnist_sgd.ipynb' notebook**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Through a simple logistic regression trained with SGD for MS and here's, the summary of where we got to, we have nearly built a module, a model module and a training loop from scratch. And we were going to kind of try and finish that and after we finished that and then going to go through this entire notebook backwards right so having gone like top to bottom, but I'm going to go back through bottom to top okay. So you know this was that little handwritten and end module class. We created, we defined our loss, we defined our learning rate and we defined our optimizer, and this is the thing that we're going to try and write by hand in a moment so that stuff that and that we're stealing with from pytorch, but that we've written Ourselves and this week written us all, so the basic idea was we're going to go through some number of epochs. So let's go through one epoch: okay and we're going to keep track of how much for each mini batch. What was the loss so that we can report it at the end, we're going to turn our training data loader into an iterator so that we can loop through it live through every mini batch, and so now we can go and go ahead and say for tensor. In the length of the data loader and then we can call next to grab the next independent variables and

the dependent variables from our data loader from that iterator. Okay, so then remember.

We can then pass the X tensor into our model by calling the model as if it was a function, but first of all we have to turn into a variable. Last week we were typing variable blog CUDA, to turn it into a variable. A shorthand for that is just the capital V. Now it's a capital T for a tensor capital B for a fever variable, that's just a shortcut in fastai, okay, so that returns our predictions. And so the next thing we needed was to calculate our loss because we can't calculate the derivatives of the loss of. U and calculate the loss, so the loss takes the predictions and the actuals okay, so the actuals again are the the Y tensor and again we have to turn that into a variable now. Can anybody remind me what a variable is and why we would want to use a variable here? I think once you turn it to variable, then it tracks it. So then you can do a backward on that. So you can yeah what sorry, when you turned a variable it it contract like this process of, like you know, as you add the function as the function starting earlier than each other, they can track it and I may need to backward on it back propagates and Those days yeah right so

3. 00:11:30

- **PyTorch tutorial on Autograd**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Right so a variable keeps track of all of the steps to get computed and so there's actually a fantastic tutorial on the pytorch website. So on the pipe torch website. There's a tutorial section and there's a tutorial there about Auto grad. Auto grad is the name of the automatic differentiation package that comes with a torch and it's a it's. An implementation of automatic differentiation, and so the variable class is really the key. The key class here, because that's the thing that makes turns a tensor into something where we can keep track of its gradients. So basically here they show how to create their variable, do an operation to a variable, and then you can go back and actually look at the grand function, which is the the function that it's keeping track of. Basically to calculate the gradient right. So as we do more and more operations to this very variable and the variable calculated from that variable, it keeps keeping track of it so later on. We can go dot, backward and then print grad and find out the gradient right, and so you notice we never defined the gradient. We just defined it as being X, plus 2 squared times 3 whatever, and it can calculate the gradient okay. So that's why we need to turn that into a variable, so L is now a variable containing the loss, so it contains a single number for this mini batch, which is the loss for this mini batch. But it's not just a number.

It's a it's a number as a variable, so it's a number that knows how it was calculated all right, so we're going to append that loss to our array just so we can get the average of it later. Basically, and now we're going to calculate the gradient. So L drop backward is a thing that says calculate the gradient. So remember when you recall the the network, it's actually calling our forward function. So that's like cap go through it forward and then backward is like using the chain rule to calculate the gradients backwards. Okay and then this is the thing we're about to write, which is update, the weights based on the gradients and the learning rate. Okay, zero grad will explain when we write this out by hand okay and so then, at the end, we can turn our validation data loader into an iterator, and we can then go through its length grabbing each X and way out of that and asking for the Score which we defined up here to be equal, to which thing did you predict, which thing was actual and so check, whether they're equal right

and then the main of that is going to be our accuracy? Okay, could you pass that over to Chen XI? What's the advantage that you found converted into iterator resident like used normal and on a Python loop or we're using a normal Python loop? So it's still them. This is a normal Python loop. So the question really is like compared to what right so, like the alternative.

Perhaps nothing here could be like we could use like a something like a list with an index. Oh okay, so you know the problem. There is the we want. As a few things, I mean one key one: is we want each time we grab a new mini batch, we want to be random, we want

4. 00:15:30

- **“Stream Processing” and “Generator Python”**
- **. “l.backward()”**
- **. “net2 = LogReg().cuda()”**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

A different different shuffled thing, so this you can actually kind of iterate from forever. You know you can look through it as many times as you like. So this is kind of idea. It's called different things in different languages, but a lot of languages. A lot like stream processing and it's this basic idea that, rather than saying I want the third thing or the ninth thing is just like. I want the next thing right, it's great for, like network programming, it's like grab the next thing from the network. It's great for UI programming. It's like we have the next event. Where there's somebody clicked a button. It also turns out to be great for this kind of numeric programming's. It's like. I just want the next batch of data. It means that the data like can be kind of arbitrarily long as we're just grabbing one piece at a time yeah. So you know I mean, and also I guess, the short answer is because it's our PI approach works. If I torch that light torches, data loaders are designed to be poured in this way and then so Python has this concept of a generator which is like an and and just a different type of generator. I want to physical gon na, be a snake generator or a computer of generator. Okay, a generator is a way that you can create a function that, as it says, behaves like an iterator so, like python, has recognized that this stream processing approach to programming is like super handy and helpful and supports it everywhere.

So, basically, anywhere that you user for in loop any way you use a list comprehension, those things can always be generators or iterators. So by programming this way we just get a lot of flexibility. I guess: does that sound about right, Terrence you're, the programming language expert. Did you do you want to grab that box, so you can hear so. Terrence actually does programming languages for a living, so we should ask him yeah I mean the short answer is what you said. You might say something about space, but in this case that all that data has to be in memory because we've got no doesn't have to be a memory so that most of the time, if we could pull a mini batch from something that most of the time With pipe torch, the mini batch will be read from like separate images spread over your disk on demand. So most of the time it's not in memory, but in in general. You want to keep his little in memory as possible at a time, and so the idea of stream processing also is great, because you can do compositions, you can pipe the data to a different machine. You can yeah yeah, the composition is great. You can grab scrap the next thing from here and then send it off to the next stream, which you can then grab it and do something else, which you guys all recognize, of course, in the command-line pipes and redirection. Yes, okay, thanks Terrence, it's a

benefit of working with people that actually know what they're talking about all right.

So, let's know take that and get rid of the optimizer okay. So the only thing that we're going to be left with is the negative log likelihood loss function which we could also replace. Actually, we have a implementation of that from scratch that you know wrote in the in the notebooks. So it's only one line of code, as we learned earlier, you can do it with a single if statement. Okay, so I don't know why I was so lazy is to include this. So what we're going to do is we're gon na again grab this module that we've written ourselves the logistic regression module we're going to have one epoch again, we're going to loop through each thing in an iterator again they're going to grab our independent and dependent variable For the mini batch again pass it into our network again calculate the loss, so this is all the same as before, but now we're going to get rid of this optimizer dot step and we're going to do it by hand. So the basic trick is, as I mentioned, we're not going to do the calculus by hand, so we call L drop backward to calculate the gradients automatically and that's going to fairly in our weight matrix. So do you remember when we created our let's go back and look at the code for here's that module we built so the weight matrix for the for the linear layer weights, recall l1 w and for the BIOS record l1 B? All right! So though they were the attributes we created, so I've just put them into things called W and B just to save some typing.

Basically so W is our weights B is our biases, and so the weights remember the weights are a variable and to get the tensor out of the variable. We have to use data all right, so we want to update the actual tensor. That's in this variable, so we say weights data minus equals, so we want to go in the opposite direction to the gradient. The gradient tells us which wears up. We want to go down whatever is currently in the gradients times the learning rate. So that is the formula for gradient descent all right. So, as you can see it's it's like as as easier thing as you can possibly imagine. It's like literally update the weights to be equal to be equal to whatever they are now minus the greater the gradients times our learning rate and do the same thing for the bias. Anybody have any questions about that step in terms of like why we do it or how it did. You have a question about that that step, but when we do the next job deal the next here. Yes, yes, so when is the end of the loop? How do you grab the next element, so this is going through each h-index in range of length. So is this going 0, 1. 2. 3. At the end of this loop, it's going to print out the mean of the validation set, go back to the start of the epoch, at which point it's going to recreate a new, a new iterator, okay.

So, basically, behind the scenes in Python, when you call it a on on this, it basically tells it like reset its state to create a new iterator and, if you're interested in how that works, the the code is all you know available for you to look at So we could look at like MD trained. Your is a fastai dot data setup model data loader. So we could like take a look at the code of that. So we could take a look at the code of that and see exactly how it's being built right, and so you can see here that here's the next function right, which basically is keeping track of how many times it's been through in this self dot. I and here's the Edith function, which is the thing that gets pretty cold when you, when you create a new iterator, and you can see it's basically passing it off to something else, which is a type data loader, and then you can check out data loader if You're interested to see how that's implemented as well um, so the data loader that we wrote basically uses multi-threading to allow it to have multiple of these going on. At the same time, it's actually a great. It's really simple. It's like it's only about a screen full of code, so if you're interested in simple Modi threaded programming, it's a good thing to look at okay. Now, oh yes, why? How do you wrap this in four epoch in range? One since that'll only run once because in real life, we would normally be running multiple so like in this case, because it's a linear model, it actually basically trains to as good as it's going to get in one he park.

So if I type three here it actually, it actually won't really improve after the first epoch. Much at all, as you can see right, but when we go back up to the top we're going to look at some slightly deeper and more interesting versions which will take more a box. So you know if I was turning this into a into a function. You know I'd be going like you know: death, train model and one of the things you would pass you and is like number of epochs kind of thing. Okay, great so one thing to remember is that when you're, you know creating these neural network layers and remember like this is just as pipe Rogers concerned. This is just a it's an end up module it could be a week could be using it as a layer who could be using to a function. We could be using it as a neural net. Pipe torch doesn't think of those as different things right. So this could be a layer inside some other network. Okay. So how do gradients work so if you've got a layer which remember is just a bunch of we can think of it, basically as its activations right or some activations, that get computed through some other nonlinear activation function or through some linear function and from that layer. We it's very likely that we're then, like, let's say putting it through a matrix product right to create some new layer, and so each one of these.

So if we were to grab like one of these activations right is actually going to be used to calculate every one of these outputs right. And so, if you want to calculate the the derivative, you have to know how this weight matrix, impacts, that output and that output and that output and that output right and then you have to add all of those together to find. Like the total impact of this, you know across all of its outputs, and so that's why in pytorch, you have to tell it when to set the gradients to zero right, because the idea is that you know you could be like having lots of different loss Functions or lots of different outputs in your next activate set of activations or whatever, all adding up, increasing or decreasing your gradients right, so you basically have to say: okay, this is a new calculation reset. Okay, so here is where we do that so before we do LDAP backward, we say reset, okay, so let's take our weights. Let's take the gradients, let's take the ten so that they point to and then zero underscore. Does anybody remember from last week what underscore does as a suffix in pytorched yeah? I have to read the language, but basically it changes. I put into place that language is in place yeah exactly so. It sounds like a minor technicality, but it's super useful to remember.

Every function, pretty much has an underscore version suffix, which does it in place yeah, so normally zero returns, a tensor of zeros of a particular size, so zero underscore means replace the contents of this. With a bunch of zeros, okay, all right, so that's that's it right! So that's like SGD from scratch and if I get rid of my menu bar, we can officially say it fits within a screen yeah. So of course we haven't got our definition of logistic regression. Here, that's another half the screen, but basically there's there's not much to it. Yes, sir, so later on, we have to do this more the gradient. Is it because you might find like a wrong minima local minima? That way, we have to kick it out and that's we have to do multiple times when the surface is getting more common. Why do you need multiple epochs? Is that your question? Well, I mean a simple way to answer. That would be, let's say our learning rate was tiny right. Then it's just not gon na get very far right. There's nothing that says going through one epoch is enough to get you all the way there so then it'd be like okay. Well, let's increase our learning rate and it's like yeah sure will increase our learning rate but who's to say that the highest learning rate that learned stabili is is enough to learn this as well as it can be learned and for most data sets for most architectures. One epoch is very rarely enough to get you to the best result. You can get too.

You know, linear models are just they're very nicely behaved, you know, so you can often use higher learning rates and learn them well quickly. Also, they they don't. You can't like

generally get as good at accuracy, so there's not as far to take them either so yeah doing one epoch is going to be the rarity all right. So let's go backwards so going backwards. We're basically going to say all right. Let's not write those two lines again and again again: let's not write those three lines again and again and again: let's have somebody do that for us right. So that's like that's. The only difference between that version in this version is rather than saying, dot zero ourselves. Rather than saying gradient times, Amara ourselves, these are wrapped up for us. Okay, there is another wrinkle here, which is this approach to updating the the weights is actually pretty inefficient. It doesn't take advantage of momentum and curvature, and so in the do course we learn about how to do momentum from scratch as well. Okay, so if we actually just use plain old SGD, then you'll see that this Alone's much slower so now that I've typed just plain old SGD here. This is now literally doing exactly the same thing as our slow version, so I have to increase the learning rate. Okay, there we go so this. This is now the same as the one we wrote by hand. So then all right, , let's do a little bit more stuff automatically.

Let's not you know, given that every time we train something we have to loop through epoch, flip through batch do forward get the loss zero. The gradient do backward, do a step of the optimizer. Let's put all that in a function, okay and that function is called fit. Alright, there it is okay, so let's take a look at fit fit go through each epoch go through each batch. Do one step keep track of the loss and, at the end, calculate the validation all right, and so then step so if you're interested in looking at this, this stuff's all inside fastai dot model, and so here is dead right, zero. The gradients calculate the loss. Remember pipe torch tends to call it criterion rather than loss, all right. Do it backward and then there's something else we haven't learned here, but we do learn the deep learning course which is gradient reading. So you can ignore that alright, so you can see now, like all the stuff that we've learnt when you look inside the actual frameworks. That's the code, you see, okay, so that's what fit does and so then the next step would be like okay. Well, this idea of like having some weights and a bias and doing a matrix product. In addition, let's put that in a function this thing of doing the log softmax, let's put that in a

5. 00:32:30

- **Building a complete Neural Net, from scratch, for Logistic Regression in PyTorch, with “nn.Sequential()”**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Function and then the very idea of like first doing this and then doing that this idea of like chaining functions together, let's put that into a function and that finally gets us to that. Okay, so sequential simply means through this function. Take the result. Send it to this function, etc, right and linear means create the weight matrix, create the biases okay. So that's that's yeah right, so we can. Then you know, as we started to talk about like turn this into any deep neural network. By saying you know, rather than sending this straight off into ten activations, let's, let's put it into say: 100 activations. We could pick whatever one number. We, like put it through a RAL you to make it nonlinear, put it through another linear layer, another rally ER and then our final output with our final activation function right, and so this is now a deep network. So we could fit that and this time now, because it's like deeper I'm actually going to run a few more a pox right and you can see the accuracy increasing okay. So if you try and increase the learning rate here, it's like 0.1 further. It actually starts to become unstable. Now

I'll show you a trick. This is called learning rate annealing and the trick is this: when you're trying to fit to a function right, you've been taking a few steps step step step as you get close to the middle like get close to the bottom, your steps probably want to become smaller Right otherwise, what tends to happen? Is you start finding you're doing this right, and so you can actually see it here right.

It got 93 94 and a bit 94 694 eight, like it's kind of starting to flatten. Now, right now, that could be because it's kind of done as well as it can, or it could be, that it's going to growing backwards and forwards. So what does a good idea, as is later on in training, is to decrease your learning rate and to take smaller steps? Okay, that's called learning rate annealing, so there's a function in fastai called set learning rates. You can pass in your optimizer and your new learning rate, and you don't see if that helps right and very often it does about about an order of magnitude in the deep learning course. We learn a much much better technique than this to do this all automatically and at a more granular level, but if you're doing it by hand, you know like an order of magnitude at a time, is what people generally do so you'll see people in papers talk About learning rate schedules, this is like a learning rate schedule, so this schedule just a moment Erica will come to us. First has got us 297. Okay and I tried kind of going further and we don't seem to be able to get much better than that. So yeah, so here we've got something where we can get 97 % accuracy. Yes Erica. So it seems like you, change the learning rate to something very small ten times smaller than we started with. So we had point one now its point: everyone yep, but that makes the whole model train really slow.

So I was wondering if you can make it so that it changes dynamically as it approaches closer to the minimum yeah pretty much yes. So so that's some of the stuff we learned in the deep planning course, these more advanced approaches, yep, so how it is different from using an immobilizer or something that that's the kind of stuff we can do. I mean you still need annealing. As I say, we do this kind of stuff in the deep learning course so for now we're just going to stick to standard SGD. I had a question about the data loading yeah. I know it's a fast day i Function, but could you go into a little bit detail of how it's creating batches, how it's done and how it's making those decisions? Sure I'd be good to ask that on Monday night, so we can talk about it in detail. In the dig learning class, but let's, let's do the quick version here so basically there's a really nice design in pytorch, where they basically say: let's, let's create a thing called a data set right and a data set is basically something that looks like a list. It has a length right, and so that's like how many images are in the data set and it has the ability to index into it like a list right. So if you had like D equals data set, you can do length D and you can do D or some index right. That's. Basically, all the data set is as far as pipe torch is concerned, and so you start with the data set. So it's like.

Okay. D3 gives you the third image you know or whatever, and so then the idea is that you can take a data set and you can pass that into a constructor for a data loader, and that gives you something which is now iterable right. So you can now say it a DL and that's something that you can call next on and what that now is going to do is if, when you do this, you can choose to have shuffle on or shuffle off shuffle on means. Give me random, mini-batch, shuffle off means drove through it sequentially and so what the data loader does now when you say next, is it basically, assuming you said, shuffle equals true it's going to grab. You know if you've got a batch size of 64 64 random integers between 0 and length and call this 64 times to get 64 different items and jam them together. So fastai uses the exact same terminology and the exact same API. We just do some of the details differently so specifically, particularly with computer vision. You often want to do a lot of pre pre processing data augmentation like flipping changing the colors a little bit rotating those turn out to be really

computationally expensive. Even just reading the JPEGs turns out to be computationally expensive, so plate or chooses an approach where it fires off multiple processors to do that in parallel, where else the faster I librarian says, does something called multi-threading, which is a much faster way of doing it. Yes, you know so I mean pork.

Is there really pork in the sense that all of the elements? So it's a shuffle at the beginning of the pork, something like that? Yeah yeah, I mean not all libraries work, the same way. Some do sampling with replacement some. Don't we actually the first a a library hands off the shuffling off to the set to the actual pipe torch version and I believe the pipes version, yeah, actually shuffles and an epoch covers everything once I believe. Okay, now the thing is when you start to get these bigger networks, potentially you're, getting quite a few parameters right. So I want to ask you to calculate how many parameters there are, but let's, let's remember here: we've got 28 by 28 input into a hundred output and then a hundred into a hundred and then 100 into 10, all right and then for each of those we've Got weights and biases, so we can actually do this net dot parameters returns a list where each element of the list is a matrix or actually a tensor of the parameters for that not just for that layer, but if it's a layer with both weights and biases, That would be two parameters right so basically returns us a list of all of the tensors containing the the parameters. Some elements in pytorch tells you how how big that is right. So, if I run this here is the number of parameters in each layer. So I've got 784 inputs and the first layer has a hundred outputs.

So therefore, the first weight matrix is of size, 78,000, 400 and the first bias vector is of size, 100, okay and then the next one is a hundred by hundred okay and there's 100 and then the next one is 100 by 10 and then there's my bias. Okay, so there's the number of elements in each layer - and I add them all up - it's nearly a hundred thousand okay and so I'm possibly at risk of overfitting yeah all right. So we might want to think about using regularization. So a really simple, common approach to regularization in all of machine learning is something called l2 regularization and it's super important super handy. You can use it with just about anything right and the basic idea anyway. So LT rotor ization. The basic idea is this: normally we'd say our loss is equal to. Let's just do our MSE to keep things kind of simple, it's equal to our predictions, our actuals, you know squared and then we sum them up. Take the average take the square root. Okay. So what if we then want to say you know what like, if I've got, lots and lots of parameters, don't use them unless they're really helping enough right like if you've got a million parameters, and you only really needed ten parameters to be useful. Just use ten. All right, so how could we like tell the loss function to do that, and so basically, what we want to say is hey. If a parameter is zero, that's no problem! It's like it doesn't exist at all. So let's penalize a parameter for not being zero right.

So what would be a way we could measure that? How can we like calculate how under Oh our parameters are I can you pass that to Qin Shi? Please Ernest. You calculates the average of positive parameters, can't quite be the average plus. Yes, Taylor yeah. Yes, you figured it out. Okay, so I think if we like, assuming all of our data, has been normalized standardized. However, you want to call it. We want to check that they're like significantly different from zero right, not the data that the parameter is rather would be significantly and the parameters don't have to be normalized or anything that is calculated right. Yes, a significantly different from zero right. I suppose I just assuming that the data has been normalized so that we can compare that money. Oh yeah, I thought of yeah right and then those that are not significantly different from zero. We can provoke. I just draw - and I think Chen she's going to tell us how to do that. You just figured it out right. The could do that. That would be called l1, which is great, so l1 would be the absolute value of the weights. Average l2 is actually the sum yeah

yeah exactly so we just take this. We can just we don't even have to square root, so we just take the squares of the weights themselves and then like. We want to be able to say like okay. How much do we want to penalize not being zero right? Because if we actually don't have that many parameters, we don't want to regularize much at all.

If we've got heaps, we do want to regularize a lot right. So then we put a parameter yeah right, except I have a role in my classes, which is never to use Greek letters. So normally people use alpha, I'm going to use hey okay, so so this is some number which you often see something around kind of 1e. Neg 6 to 1 enoch, 4 ish right now. We actually don't care about the loss. When you think about it, we don't actually care about the loss other -- than like, maybe to print it out or we actually care about, is the gradient of the loss. Okay, so the gradient of that right is that right. So there are two ways to do this: we can actually modify our loss function to add in this square penalty or we could modify that thing where we said weights equals weights minus gradient times, learning rate to subtract that as well right actually to add that as Well - and these are roughly, these are kind of basically equivalent, but they have different names. This is called l2 regularization right. This is called weight decay. So in the neural network, literature you know that version kind of was the how it was first posed in the neural network. Literature where else this other version is kind of how it was posed in the statistics, literature and yeah. You know they're they're equivalent, as we talked about in the deep learning class.

It turns out they're not exactly equivalent, because when you have things like momentum and atom, it can behave differently and two weeks ago a research figured out a way to actually do proper weight. Decay in modern optimizers and one of our first day, students just implemented that in the first day I library so first a is now the first library to actually support this properly so anyways. So for now, let's do the the version which applied torch calls weight decay, but actually it turns out, based on this paper two weeks ago, is actually l2 regularization. It's not quite correct, but it's close enough so here we can they weight decay as one in x3. So it's going to set our constant, our penalty, multiplier a21 in X 3, and it's going to add that to the loss function, okay and so let's make a copy of these cells just so we can compare hope. This actually works. Okay and we'll set this running. Okay, because there's now optimizing well, except if you actually so, I've made a mistake here, which is, I didn't rerun this cell. This is an important thing to kind of remember, since I didn't run this rerun this cell here when it created the optimizer and said net dot parameters. It started with the parameters that I had already trained right, so I actually hadn't recreated my network okay. So, actually you go back and rerun this cell first to recreate the network then go through and run this okay.

There we go. So, let's see what happens so you might notice them notice something kind of kind of counterintuitive here, which is that that's our training error right now. You would expect our training error with regularization to be worse. That makes sense right because we're like we're penalizing parameters that specifically, can make it better, and yet actually it started out better not worse, so why could that be? So the reason that can happen is that if you have a function that looks like that right, it takes potentially a really long time to train. Where else, if you have a function, that kind of looks more like that. It's going to train a lot more quickly and there are certain things that you can do, which sometimes just like can take a function, that's kind of horrible and make it less horrible and it's sometimes weight decay. You can actually make your functions. A little more nicely behaved, and that's actually happened here so, like I just mentioned that to say like don't let that confuse you right, like white decay, really does penalize the training set and look so strictly speaking. The final

number we get to for the training set shouldn't end up, be beat being better, but it can train sometimes more quickly. All right! Yes, can you pass it a chance? You don't get it okay. Why making it faster like Zee Time Matters like the training time, memories? No, it's just after one epoch right. So after one epoch.

Now, congratulations for saying I don't get it! That's like the best thing anybody can say you know so hopeful. This here was our training. Without wait, okay, okay and this here is our training with wait: okay, okay, so this is not really just a time. This is related to just an epoch right after one Apoc. My claim was that you would expect the training set all other things being equal to have a worse loss with weight decay because we're penalizing it. You know this has no penalty. This has a penalty, so the thing with the penalty should be worse and I'm saying oh, it's not that's weird right, and so the reason it's not is because in a single epoch it matters a lot as to whether you're trying to optimize something that's very bumpy Or whether you're trying to optimize something that's kind of nice and smooth, if you're, trying to optimize something, that's really bumpy, like imagine in some high dimensional space right, you end up kind of rolling around through all these different tubes and tunnels and stuff. You know where else, if it's just smooth, you just go boom item, it's like imagine, of marble rolling down a hill where one of them you've got like it's a called Lombard Street in San Francisco. It's like backwards forwards backwards forwards. It takes a long time to drive down the road right.

Where else you know, if you kind of, took a motorbike and just went straight over the top, you just make sure boom right, so so weather so kind of the shape of the loss function. Surface. You know, impacts or kind of defines how easy it is optimized and therefore, how far can it get in a single epoch and based on these results, it would appear that weight decay here has made it this function easier to optimize, so just to make sure it's The penalizing is making the optimizer more than likely to reach the global minimum. No, I wouldn't say that my claim actually is that at the end, it's probably going to be less good on the training set, and indeed this does look to be the case. At the end, after five epochs, our training set is now worse with weight decay. Now, that's what I would expect right. I would expect like, if you actually find like, I never used a term global optimum, because it's just not something we have any guarantees about. We don't really care about, or just care like, where do we get to after a certain number of epochs. We hope that we found somewhere, that's like a good solution, and so by the time we get to like a good solution, the training set with weight decay. The loss is worse because it's very right, but on the validation set, the loss is better right because we penalized the training set in order to kind of try and create something it generalizes better. So we've got more parameter.

You know that the parameters that are kind of pointless in l0 and it generalizes better right so so all we're seeing is that it just got to a good point after one epoch is really orbiting. Obviously, no, no I, but if you buy, if you mean just weight decay, you always make the function surface smoother. No, it's not always true, but it's like it's worth, remembering that if you're having trouble training of function, adding a little bit of weight, decay may may help what so by analyzing the parameters, what it does as its moons out the loss function. I mean it's not it's not why we do it. You know the reason why we do. It is because we want to penalize things that aren't zero. To say, like don't, make this per a high number, unless it's really helping the Lasser lodge right, set it to 0, if you can, because setting as many parameters to zero as possible means that it's going to generalize better right, it's like the same as having a Smaller network, okay, so that's that's! We do that's why we do it, but it can change how it learns as well. So let's, okay, this one moment, okay, so I just wanted to check how we

actually went here so after the second epoch yeah. So you can see here, it's really has helped right after the second epoch, before we got to 97 % accuracy, now we're nearly up to about 98 % accuracy right and you can see that the loss was 0.08 versus 0.13 right.

So adding regularization has allowed us to find a you know: three percent versus two percent, so like a fifty percent, better solution, yes Erica. So there are two pieces to this right. What is l2 regularization and the way to key know they're the same, so my claim was they're the same thing right. So white decay is the version. If you just take the derivative of LT regularization, you get weight decay, so you can implement it either by changing the loss function with an with a squared loss penalty or you can implement it by adding the weights themselves as part of the the gradient okay yeah. It's just gon na finish the questions. Yes, okay, pass it to division. Can we use the regularization convolution, absolutely so a compilation layer just is: is White's aim? Jeremy? Can you explain why you thought you needed weight decay in this particular problem, not easily? I mean other than to say it's something that I would always try speaking well yeah I mean okay. So even if I yeah okay, that's a good point unit. So if if my training loss was higher than my validation loss, then I'm under fitting right. So, there's definitely no point recognizing right if, like that, would always be a bad thing. That would always mean you need like more parameters in your model. In this case, I'm I'm overfitting that doesn't necessarily mean regularization will help, but it's certainly worth trying. Thank you. You know that's a great point.

There's one more question: yep Tyler doing a pass over there. So how do you choose the optimal number of epic? You do my take learning cause it's a it's! That's a long story and a lot to lots of users by here on here, ollie, it's a bit of both. We just don't. As I say, we don't have time to cover best practices in this class, we're going to learn the kind of fundamentals yeah okay. So, let's take a six minute break and

6. 00:58:00

- **Fitting the model in 'lesson4-mnist_sgd.ipynb' notebook**
- **The secret in modern ML (as covered in the Deep Learning course): massively over-paramaterized the solution to your problem, then use Regularization.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Come back at 11:10, alright, so something that we cover in great detail in the deep learning course. But it's like really important to mention here is that is it the secret? In my opinion, to kind of modern machine learning techniques is to massively over parameterize the solution to your problem. Right like as we've done here, you know, we've got like a hundred thousand weights when we only had a small number of 28 by 28 images and then use regularization. Okay, it's like the direct opposite of how nearly all statistics and learning was done for decades before and still most kind of like senior lecturers at most universities in most areas of have this background, where they've learned the correct way to build a model is to like Have as few parameters as possible right - and so hopefully, we've learnt two things so far. You know one is: we can build very accurate models even when they have lots and lots of parameters like a random forest has a lot of parameters - and you know this here. Deep network has a lot of parameters and they can be accurate right and we can do that by either using bagging or by using regularization. Okay and regularization in neural nets means either weight decay, also known as kind of filter, regularization or drop out, which we won't worry

too much about yeah.

So like it's a it's a very different way of thinking about building useful models and like I just wanted to kind of warn you that once you leave this classroom like even possibly when you go to the next faculty members, talk like there'll, be people at USF As well, who entirely trained in the world of like models with small numbers of parameters, you know your next boss is very likely to have been trained in the world of models with small numbers of parameters, the idea that they are somehow more pure or easier or Better or more interpretable or whatever I I am convinced that that is not true, probably not ever true, certainly very rarely true, and that actually models with lots of parameters can be extremely interpretive. All as we learn from our whole lesson of random forest interpretation, you can use most of the same techniques with neural nets, but with neural nets are even easier right. Remember how we did feature importance by randomizing a column to see how it changes in that column. Would impact the output? Well, that's just like a kind of dumb way of calculating its gradient. How much does varying this import change the output with a neural net? We can actually calculate its gradient right so with pytorch, you can actually say what's the gradient, that the output with respect to this column, okay, you can do the same kind of thing to do partial dependence plot with an Anette - and you know I mentioned - for Those of you interested in making a real impact nobody's written basically any of these things for neural nets right, so that that that whole area needs like libraries to be written blog posts to be written.

You know some papers have been written, but only in very narrow domains like computer vision, as far as I know, nobody's written the paper saying: here's how to do structured data neural networks. You know, interpretation methods, so it's a really exciting big area. So what we're going to do, though, is we're going to start with applying this with a simple

7. 01:02:10

- **Starting NLP with IMDB dataset and the sentiment classification task**
- **NLP = Natural Language Processing**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Linear model - this is mildly terrifying for me, because we're going to do NLP and NLP faculty expert is in the room, so David just yell at me. If I screw this up too badly and so NLP refers to, you know any any kind of modeling where we're working with with natural language tests right and it. Interestingly enough, we're going to look at a situation where a linear model is pretty close to the state of the art for solving a particular problem. It's actually something where I actually surpassed this baited state of the art in this using a recurrent neural network. A few weeks ago, but this is actually going to show you pretty close to the state of art with with a linear model, we're going to be working with the IMDB. I am DVD data set. So this is a data set of movie reviews. You can download it by

8. 01:03:10

- **Tokenizing and 'term-document matrix' & "Bag-of-Words' creation**
- **"trn, trn_y = texts_from_folders(f'{PATH}train', names)" from Fastai library to build arrays of reviews and labels**

- **Throwing the order of words with Bag-of-Words !**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Following these steps, and once you download it, you'll see that you've got a train and a test directory and in your train directory you'll, see, there's a negative and a positive directory and in your positive directory, you'll see, there's a bunch of text files and here's an Example of a text file, so somehow we've managed to pick out a story of a man who has a natural feelings for a pig as our first choice that wasn't intentional, but it'll be fun. So we're going to look at these movie reviews and for each one we're going to look to see whether they were positive or negative. So they've been put into one of these folders. They were downloaded from from IMDB the movie database and review site, the ones that were strongly positive, went positive, strongly negative, went negative and the rest they didn't label at all, so there's only highly polarized reviews. So, in this case you know we have an insane violent mob, which, unfortunately it is too absurd, too off-putting those in the area be turned off, so the label for this was a zero which is negative okay. So this is a negative review, so in the fastai library, there's lots of little functions and classes to help with most kinds of domains that you do machine learning on for NLP. One of the simple things we have is texts from folders there's just going to go ahead and go through and find all of the folders in here with these names and create a labeled data set, and you know don't let these things ever. Stop you from understanding.

What's going on behind the scenes, okay, we can grab its source code and, as you can see, it's tiny. You know it's like five lines. Okay, so I don't like to write these things out in full. You know, but hide them behind at all functions. So you can reuse them, but basically it's going to go through each directory and then within that, so I go through yeah, go through each directory and then go through each file in that directory and then stick that into this array of texts and figure out what Folder it's in and stick that into the array of labels. Okay, so that's how we basically end up with something where we have an array of the reviews and an array of the labels. Okay, so that's our data, so our job will be to take that and to predict that okay and the way we're going to do it is we're gon na throw away like all of the interesting stuff about language, which is the order in which the words are In right now, this is very often not a good idea, but in this particular case it's going to turn out to work like not too badly. So let me show you what I mean by like throwing away the order of the words like normally the order of the words matters a lot if you've got a not before something, then that not refers to that thing right so, but the thing is when, in This case we're trying to predict whether something is positive or negative.

If you see the word absurd appear a lot right, then maybe that's a sign that this isn't very good. So you know cryptic, maybe there's a sign that it's not very good. So the idea is that we're going to turn it into something called a term document matrix where for each document, are you each review, we're just going to create a list of what words are in it, rather than what order they're in so. Let me give an example: can you see this okay? Okay, so here are four movie reviews that I made up. This movie is good. The movie is good they're. Both positive. This movie is bad. The movie is bad they're, both negative right. So I'm going to turn this into a term document matrix. So the first thing I need to do is create some in-court of vocabulary. A vocabulary is a list of all the unique words that appear. Okay, so here's my vocabulary. This movie is good, the bad. That's all the words okay, and so now I'm going to take each one of my movie reviews and turn it into a vector of which words appear, and how often do they

appear right, and in this case none of my words appear twice. So this movie is good. Has those four words in it? Where else this movie is bad, has those four words in it? Okay, so this is called a term document matrix alright and this representation. We call a bag of words, representation right, so this here is a bag of words, representation of the view of the review. It doesn't contain the order of the text anymore. It's just a bag of the words.

What words are in it. It contains bad is movie this okay. So that's the first thing we're going to do is we're going to turn it into a bag of words, representation, and the reason that this is convenient for linear models is that this is a nice rectangular matrix that we can like do. Math on okay, and specifically, we can do a logistic regression and that's what we're going to do is we're going to get to a point. We do a logistic regression before we get there, though we're going to do something else, which is called naive, Bayes, okay, so scikit-learn has something

9. 01:08:50

- **sklearn "CountVectorizer()"**
- **"fit_transform(trn)" to find the vocabulary in the training set and build a term-document matrix.**
- **"transform(val)" to apply the same transformation to the validation set.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Which will create a term document matrix for us? It's called count: vectorizer, okay. So what does use it now in NLP? You have to turn your text into a list of words and that's called tokenization. Okay and that's actually non-trivial because like if this was actually, this movie is good dot right or if it was. This movie is good like how do you deal with like that punctuation? Well, perhaps more! Interestingly, what if it was this movie, isn't good, alright, so how you turn a piece of text into a list of tokens is called tokenization right and so a good tokenizer would turn this movie isn't good into this. This base quote movie. Space is space and good space right, so you can see in this version here. If I now split this on spaces, every token is either a single piece of punctuation or like this suffix and is considered like a word right, that's kind of like how we would probably want to tokenize that piece of text, because you wouldn't want good to be Like an object right because there's no concept of good right or double-quote movie is not like an object right, so tokenization is something we hand off to a tokenizer. First, AI has a tokenizer in it that we can use. So this is how we create our term document matrix with a tokenizer s. Kaitlyn has a pretty standard API, which is nice, I'm sure you've seen it a few times now before so once we've built some kind of model think we can kind of think of this.

As a model just ish, this is just defining what it's going to do, but you can call fit transform to do that right. So, in this case, fit transform is going to create the vocabulary. Okay and create the term document matrix based on the training set. Transform is a little bit different that says, use the previously fitted model, which in this case means use the previously created vocabulary. We wouldn't want the validation set in the training set to have. You know the words in different orders in the matrices right, because then they'd like to have different meanings, so this is here saying: use the same vocabulary to create a bag of words for the validation set. Could you pass that back in please what if the violation set has different set of words other than training yeah? That's a great question! So, generally most of these kind of vocab, creating approaches will have a special token for unknown. Sometimes you can you'll

also say like hey if a word appears less than three times call it unknown, but otherwise it's like, if you see something you haven't seen before, call it unknown. So that would just become a column in the bag of words is good question. All right so when we create this term document matrix the training set, we have 25,000 rows because there are 25,000 movie reviews and there are 70 5132 columns.

What does that represent? What does that mean? There are seven hundred and seventy-five thousand hundred thirty-two. What can you pass that to device

10. 01:12:30

- **What is a 'sparse matrix' to store only key info and save memory.**
- **More details in Rachel's "Computational Algebra" course on Fastai**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Just a moment: okay power sector diversion locality, yeah come on. What do you mean so like the number of words Union or the number of words that the number of unique words yeah exactly good? Okay, now most documents don't have most of these 75,000 words right. So we don't want to actually store that as a normal array in memory, because it's going to be very wasteful, so instead we store it as a sparse matrix, okay and what a sparse matrix does. Is it just stores it as something that says whereabouts of the non-zeros right? So it says, like okay term number, so document number one word number four appears and it has four of them. You know document one term number 123 has that that appears and it's a one right and so forth. That's basically how it's done, there's actually a number of different ways of storing and if you do, Rachel's, computational, linear algebra course. You'll learn about the different types and why you choose them and how to convert and so forth, but they're all kind of something like this right and you don't really on the whole have to worry about the details. The important thing to know is it's: it's efficient. Okay, and so we could grab the first review all right, and that gives us 75,000 long, sparse one long one row long matrix, okay, with 93 stored elements, so in other words, 93 of those words, are actually used in the first document.

Okay, we can have a look at the vocabulary by saying vectorizer, docket fetcher feature names that gives us the vocab and so here's an example of a few of the elements of feature names. I didn't intentionally pick the one that had Ozzie, but you know that's the important words. Obviously I haven't you the tokenizer here, I'm just bidding on space, so this isn't quite the same as what the vectorizer did, but to simplify things. Let's grab a set of all the lowercased words by making it a set, we make them unique. So this is roughly the list of words that would appear right and that length is 91, which is pretty similar to 93, and just the difference will be that I didn't use a real tokenizer yeah right. So that's, basically all that's been done there. It's probably created this unique list of words and mapped them. We could check by calling vectorizer cavalry underscore to find the ID of a particular word. So this is like the reverse map of this one right. This is like integer. Two word here is word two integer, and so we saw absurd appeared twice in the first document. So, let's check train term dark zero, comma one, two nine seven there it is - is two right or else unfortunately, Ozzie didn't appear in the unnatural relationship with a pig movie, so zero comma five thousand is zero. Okay. So that's that's our term document matrix.

Yes, so does it care about the relative relationship between the words as in the ordering of the

words no we've thrown away the orderings? That's why it's a bag of words and I'm not claiming that this is like necessarily a good idea. What I will say is that, like the vast majority of NLP work, that's been done over the last few decades generally uses this representation, because we didn't really know much better nowadays, increasingly we're using recurrent neural networks instead, which we'll learn about in our last deep learning. Lesson of part one, but sometimes this

11. 01:16:40

- **Using "Naive Bayes" for "Bag-of-Words" approaches.**
- **Transforming words into features, and dealing with the bias/risk of "zero probabilities" from the data.**
- **Some demo/discussion about calculating the probabilities of classes.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Representation works pretty well, and it's actually going to work pretty well in this case. Okay, so in fact you know most like back when I was at first me or my email company, a lot of the spam filtering. We did used this next technique, naivebayes, which is as a bag of words approach just kind of like you know, if you're getting a lot of email containing the word viagra and it's always been a spam and you never get email from your friends talking about viagra, Then it's very likely something that says by Agra, regardless of the detail of the language, is probably from a spammer all right. So that's the basic theory about like classification using a term document matrix okay. So let's talk about naive, Bayes and here's the basic idea we're going to start with our term document matrix right, and these first two is our corpus of positive reviews. These next two is our corpus of negative reviews, and so here's our whole corpus of all reviews. So what I could do is now to create a probability. I've got to call the as we tend to call these more generically features rather than words right. This is a feature movie as a feature is as a feature right. So it's kind of more now like machine learning, language, a column is a feature. Look all those we call those earth in naive Bayes. So we can basically say the probability that you would see the word this, given that the class is 1, given that it's a positive review is just the average of how often do you see this in the positive reviews right now, we've got to be a bit Careful, though, because if you never ever see a particular word in a particular class right, so if I've never received an email from a friend that said viagra right, that doesn't actually mean the probability of a friend.

Send me sending me an email about viagra is zero. It's not really zero right. I I hope I don't get an email. You know from Terrence tomorrow saying like Jeremy, you probably could use this. You know effortless meant for viagra, but you know it could happen. You know I'm sure it'd be in my best interest yeah. So so what we do is we say actually what we've seen so far is not the full sample of everything that could happen. It's like a sample of what's happened so far, so let's assume that the next email you get actually does mention viagra and every other possible word right. So basically we're going to add a row of ones. Okay, so that's like the email that contains every possible word. So that way, nothing's ever infinitely and unlikely yeah. So I take the average of all of the times that this appears in my positive corpus, plus the ones okay. So that's like the the probability that feature equals. This appears in a document given that class equals one, and so not surprisingly, here's the same thing for probability that this feature this appears given class equals zero all right same calculation, except for the zero rows, and obviously these are the same because this appears twice in The

positives, sorry, once in the positives and once in the negatives, okay, let's just put this back to what it was all right, so we can do that for every feature for every class right. So our trick now is to basically use Bayes rule to kind of fill this in.

So what we want is the probability that, given that I've got this particular document, so somebody sent me this particular email or I have this particular IMDB review. What's the probability that its class is equal to, I don't know positive right so for this particular movie review. What's the probability that it's plus is positive right, and so we can say: well that's equal to the probability that we got this particular movie review, given that its class is positive, multiplied by the probability that any movie reviews plus is positive, divided by the probability of Getting this particular movie review all right, that's just basis rule okay, and so we can calculate all of those things. But actually what we really want to know is: is it more likely that this is plus 0 or plus 1 right? So what if we actually took probability, that's plus 1 and divided by probability, that's plus 0? What if we did that right and so then we could say like okay, if this number is bigger than 1, that it's more likely to be class 1. If it's smaller than 1, it's more likely to be class 0 right. So in that case, we could just divide this whole thing right by the same version for class 0 right, which is the same as multiplying it by the reciprocal, and so the nice thing is now that's going to put a probability D on top here, which we Can get rid of right and a probability of getting the data given zero down here and the probability of getting class zero here right, and so, if we, basically what that means is we want to calculate the probability that we would get this particular document, given that The class is 1 times the probability that the class is 1 divided by the probability of getting this particular document.

Given the classes to 0 times the probability that the class is 0. So the probability that the class is 1 is just equal to the average of the labels write probability. The class is 0, is just 1 minus that right. So so there are those two numbers right. I've got an equal amount of both, so it's both 0.5. What is the probability of getting this document, given that the class is 1? Can anybody tell me how I would calculate that? Can somebody pass that so remember, so it's going to be for a particular document. So, for example, would be saying like what's the probability that this review is positive, all right, so what so you're on the right track. But what we have to gon na have to do is going to have to say: let's just look at the words it has and then multiply. The probabilities together for class equals 1 right. So the probability that a class 1 review has this is 2/3. The probability it has movie is one is, is 1 and good is 1, so the probability it has all of them is all of those multiplied together, kinda and the kinder phyla. Why is it not really, can you press it in Taylor?

12. 01:25:00

- **Why is it called “Naive Bayes”**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So glad you look horrified and skeptical where choices, not independence. Thank you. So nobody can call Tyler naive, because the reason this is naive, Bayes is because this is what happens if you take bayes's theorem, Xander naive. Why and Tyler is not naive. Anything better right! So naive, Bayes says: let's assume that if you have this movie is bloody stupid. I hate it, but the probability of hate is independent. Of the probability of bloody is an independent of the probability of stupid right, which is definitely not true right and so naive.

Bayes ain't actually very good, but I'm kind of teaching it to you, because it's going to turn out to be a convenient piece for something we're about to learn later. It's okay right! I mean it's, it's it's! I would never. I would never choose it like. I don't think it's better than any other technique, that's equally fast and equally easy, but you know it's the thing you can do and it's certainly going to be a useful foundation. So so here is now calculation right of the probability that this document is that we get this particular document, assuming it's a positive review, here's the probability given us a negative and here's the ratio and this ratio is above one. So we're going to say. I think that this is probably a positive review. Okay, so that's the Excel version, and so you can tell that I let your net touch this, because it's got latex in it. We've got actual math, so so here is.

The here is the same thing. The log count ratio of H, feature FH would if - and so here it is written out as Python okay, so our independent variable is our term document matrix. A dependent variable is just the labels of the way so using numpy. This is going to grab the rows where the dependent variable is one okay, and so then we can sum them over the rows to get the total word count for that feature across all the documents right plus one all right, because that's the email Terrance is totally Going to send me something about Viagra today, I can tell that's that's that yeah okay, so do the same thing for the negative reviews right and then, of course, it's nicer to take the log right, because if we take the log, then we can add things together. Rather than multiply them together - and once you like multiply enough of these things together, it's going to get kind of so close to zero. That you'll probably run out of floating-point right. So we take the log of the ratios and then we come. As I say, we then multiply that or a log we subtract that from the so you add that to the ratio of the class, the whole plus probabilities all right. So in order to say for each document multiply the phase probabilities by the accounts. We can just use matrix model, plane, okay and then to add on the the log of the class ratios. We can just use plus B, and so we end up with something that looks a lot like our logistic regression right, but we're not learning anything right, not in kind of assess, GED point of view we're just we're calculating it using this theoretical model, okay, and so, As I said, we can then compare that as to whether it's bigger or smaller than zero, not one anymore, because we're now in log space right and then we can compare that to the mean - and we say: okay - that's 80 %. Accurate 81 was inaccurate right.

So naive, Bayes, you know is not it's not nothing. It gave us something. Okay, it turns out that this version, where we're actually looking at how often a word appears like absurd, appeared twice. It turns out, at least for this problem, and quite often it doesn't matter whether absurd appeared twice or once all that matters is that it appeared. So what what people tend to try doing is to say take the two other term document matrix and go dot sine dot sign replaces anything positive as one and anything negative with negative one. We don't have any negative counts. Obviously so this buying arises it so it says it's, I don't care that you saw absurd twice and it's care that you saw it right. So if we do exactly the same thing with the binarized version, then you get a better result. Okay, okay. Now

13. 01:30:00

- **The difference between theory and practice for “Naive Bayes”**
- **Using Logistic regression where the features are the unigrams**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

This is the difference between theory and practice. Right in theory, naive Bayes sounds okay, but it's it's naive, unlike Tyla, it's naive right, so what Tyler would probably do would instead say rather than assuming that I should use these coefficients. Ah, why don't we learn them? So it sound reasonable, Tyler, yeah, okay, so let's learn them so we can. You know we can totally learn them. So let's create a logistic regression right and let's fit some coefficients and that's going to literally give us something with exactly the same functional form that we had before. But now, rather than using a theoretical R and a theoretical B. We're going to calculate the two things based on logistic regression and that's better okay. So so it's kind of like yeah. Why? Why do something based on some theoretical model, because theoretical models are never gon na, be as accurate, pretty much as a data-driven model right because theoretical models unless you're dealing with some - I don't know like physics, thing or something where you're like okay? This is actually how the world works. There really is, no, I don't know we're working in a vacuum, and this is the exact gravity and blah blah blah right, but most of the real world. This is how things are like it's better to lie on your coefficients and calculate them.

Yes, you know generally, what's this do liquid through, I was hoping you'd ignore, not notice, but or basically in this case, our term document matrix is much wider than it is tall. There is a reformulation of mathematically, basically almost a mathematically, equivalent reformulations of logistic regression. That happens to be a lot faster when it's wider than it is tall. So the short answer is, if you don't put that here anytime, it's wider than it is tall. What really comes true it'll run. This runs in like two seconds. If you don't have it here, it'll take a few minutes so like in math. This is kind of concept of dual versions of problems which are kind of like equivalent versions that sometimes work better for certain situations. Okay here is so here is the binarized version. Okay and it's it's about the same right, so you can see I've fitted it with the the sine of the dock of the dr. Murdock matrix and predicted it with this right now. The thing is that this is going to be a coefficient for every term, where I was about 75,000 terms in their vocabulary, and that seems like a lot of coefficients given that we've only got twenty five thousand reviews. So maybe we should try regularizing this, so we can use regularization built into SK. Learns logistic regression class, which is C, is the parameter that they use. A small process is slightly weird, as smaller parameter is more regularization right.

So that's why I used wanting eight to basically turn off regularization yeah. So if I turn on regularization set it to 0.1, then now it's 88 %, okay, which makes sense. You know you, wouldn't you would think like 75,000 parameters for 25,000 documents. You know it's likely to overfit; indeed it did overfit, so this is adding l2 regularization to avoid overfitting. I mentioned earlier that as well as l2, which is looking at the weight squared, there's also l1, which is looking at just the absolute value of the way right. I I was kind of pretty sloppy in my wording before I said that l/2 who tries to make things zero, that's kind of true, but if you've got two things that are highly correlated, then l2 regularization will make move them both down together. It won't make one of them zero and one of them nonzero right so l1 regularization actually has the property that it'll try to make as many things zero as possible. Where else l2 regularization has a property that it tends to try to make kind of everything smaller. We actually don't care about that difference in really any modern machine learning, because we very rarely try to directly interpret the coefficients. We try to understand our models through interrogation using the kind of techniques that we've learned. The reason that we would care about l1 versus l2 is simply like which one ends up with a better error on the validation set. Okay - and you can try both with scikit-learn logistic regression l2 actually turns out to be a lot faster because you can't use your equals true unless you have L 2, so you know and l2 is the default.

So I didn't really worry too much about that difference. Yet so you can see here if we use regularization and binarized, we actually do pretty well. Okay! So yes, can you pass that back to W please before we learned about elastic net right like combining l1 and l2 yeah yeah yeah? You can do that, but I mean it's like you know, with with deeper models and yeah I've never seen anybody find that useful. Okay, so the last thing

14. 01:35:40

- **Using Bigram & Trigram with Naive Bayes (NB) features**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

I'll mention is that you can, when you do your count, vectorizer order. That was when you do your account vectorizer. You can also ask for n grams right by default, we get uni grams. That is single words, but if we, if we say Engram range, equals 1, comma 3, that's also going to give us by grams and trigrams, by which I mean. If I now say: okay, let's go ahead and do the count. Vectorizer cat feature names now. My vocabulary includes a bigram right, bypassed by vengeance and a trigram by vengeance, 5 euro miles right. So this is now doing the same thing. But after tokenizing it's not describing each word and saying that's part of our vocabulary: two words next to each other and each three words next to each other and this 10. This turns out to be like super helpful in like taking advantage of bag of word approaches, because we now can see like the difference between, like you know, not good versus, not bad. This is not terrible right or even like double quote. Good double quote, which is probably going to be sarcastic right, so using trigram features, actually is going to turn out to make both naivebayes and logistic regression quite a lot better. It really takes us quite a lot further and makes them quite useful. I have a question about the token icers, so you are saying some marks features. So how are these diagrams and trigrams selected right? So since I'm using a linear model, I didn't want to create too many features.

I mean it actually worked. Fine, even without max features. I think I had something like I can't remember: 70 million coefficients it still worked right, but just there's no need to have 70 million coefficients. So if you say max features equals 800,000, the count vectorizer will sort the vocabulary by how often everything appears whether it be uni ground by graham diagram, and it will cut it off after the first 800,000. Most common engrams Engram is just a generic word for uni gram by Graham and trigram. So that's why the train term doctype is now 25 thousand by 800,000 and like if you're, not sure what number this should be. I just picked something that was really big and you know didn't didn't worry about it too much and it seemed to be fine, like it's not terribly sensitive, all right, okay! Well, that's we're out of time. So what we're going to see next week and by the way you know we could have replaced this logistic regression with our pytorch version and next week, we'll actually see something in the fast, a library that does exactly that. But also what we'll see next week. Starting next week, tomorrow is how to combine logistic regression and naive Bayes. Together, you get something that's better than either and then we'll learn how to move from there to create a deeper neural network to get a pretty much state-of-the-art result for structured learning all right.

So we'll see you then

Lesson 11

Outline

- Rewriting fit from scratch
- Digression of Momentum
- Rewriting gradient and step within fit function
- NLP
- Bag of words / CountVectorizer
- LogisticRegression w. Sentiment

Video Timelines and Transcript

1. 00:00:01

- **Review of optimizing multi-layer functions with SGD**
- **“d(h(g(f(x)))) / dw = 0,6”**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So, let's just yeah: let's start by reviewing kind of what we've learnt about optimizing, multi-layer functions with SGD, and so the idea is that we've got some data and then we do something to that data. For example, we multiply it by a weight matrix and then we do something to that. For example, we put it through a softmax or a sigmoid, and then we do something to that, such as do a cross, entropy loss or a root mean squared error, loss. Okay and that's gon na like give us some scalar, so this is gon na, have no hidden layers. This has got a linear layer, a nonlinear activation being a soft Max and a loss function being a root mean squared error or a cross entropy all right and then we've got our input: data, port, linear, nonlinear bus. So, for example, if this was sigmoid or softmax, and this was cross entropy, then that would be logistic regression. So it's still ya cross entropy yeah. Let's do that next short for now. Think of it, like think of RIT, means great error. Same thing: some loss function. Okay, now we'll look at cross entropy again in the moment. So how do we calculate the derivative of that with with respect to our weights right? So really, it would probably be better if we said X, comma, W yeah, because it's really a function of the weights as well, and so we want the derivative of this with respect to our weights. Sorry, I put it in the wrong spot.

Oh G, f of X, comma W! That's why that didn't make sense all right, so to do that we just basically we do the chain rule. So we just say that this is equal to H of U and u equals G! Well, G du equals G of V and V equals f of X, so we can just rewrite it like that right and then we can do the chain rule. So we can say that's equal to H. The derivative is H: u, by G dash V by F, dash X hacker with all that so far, okay. So, in order to take the derivative with respect to the weights, therefore, we just have to calculate that derivative with respect to W using that exact formula. So if we had in there yeah yeah so so d of all that DW would be yep. So then, if, if we, you know, went further here and had like another linear layer right, that's going to

be more room now that linear layer I cover w2 okay. So we have another Lydia lair. There's no difference to now calculate the derivative with respect to all of the parameters. We can still use the exact same chain rule right, so so don't think of the multi-layer network as being like things that occur at different times. It's just a composition of functions, and so we just use the chain rule to calculate all the derivatives at once. You know there's that there just a set of parameters that happen to appear in different parts of the function, but look at that. The calculus is no.

No different so to calculate this with respect to w1 and w2. You know it's it's just you just increase, you know W. You can just now just call it W and say w1 is all of those weights. So the result. Ah, that's a great question. So what you're going to have, then, is a list of parameters right so here's w1 and like it's it's it's probably some kind of higher rank tensor. You know like if it's a convolutional layer it'll, you don't be like a wreck, three tensor or whatever, but we can flatten it out. That would just make it a list of parameters. There's w1. Here's w2. Okay! It's just another list of parameters right and here's. Our loss, which is a single you, know a single number. So therefore, our derivative is just a vector of that same length right it's. How much does changing that value of W affect the loss? How much does changing that value of W affect the loss right? So you can basically think of it as a function like you know, y equals ax, 1, plus BX, 2, plus C right and say like oh, what's the derivative of that with respect to a B and C, and you would have three numbers: the derivative with respect To a and B and C - and that's all, this is right, if the derivative with respect to that weight, that weight and that weight and that weight that went that way to get there inside the chain rule.

We had to calculate and a lot of detail here, but we had to calculate like jacobians so like the derivative, when you take a matrix product, is you've now got something where you've got like a a weight. Matrix and you've got an input vector. These are the activations and the previous layer right and you've got some new output, activations right and so now you've got to say like okay, for this particular sorry for this particular weight. Hablas changing this particular weight change, this particular output and how does changing this particular weight change, this particular output and so forth? So you kind of end up with these higher dimensional tensors showing like for every weight. How does it affect every output all right, but then, by the time you get to the last function, the last function is going to have like a mean or a sum or or something so they're all going to get added up in the end, and so this Kind of thing, like I don't know it drives me a bit crazy to try and calculate it out by hand or even think of it, step by step, because you tend to have like you just have to remember for every input and Aleya for every output. In the next layer you know you're going to have to account for every weight for every output you're going to have to have a separate grant. Yet one good way to look at this is to learn to use pytorches like dot grad attribute and got backward method manually and like look up the to tour, the pytorched tutorials, and so you can actually start setting up some calculations with a vector import In the vector output and then type dot backward and then say type grad and like look at it right and then do some really small ones, with just two or three items in the input and output vectors and let make the make the operation like plus two Or something and like see what the shapes are make sure it makes sense yeah, because it's kind of like this vector matrix calculus is not like introduces zero new concepts to anything you learnt in high school, like strictly speaking, but getting a feel for how these shapes Move around I find took a lot of practice. You know the good news.

Is you almost never have to worry about it? Okay, so we were talking about. Then

2. 00:09:45

- **Review of Naive Bayes & Logistic Regression for NLP with lesson5-nlp.ipynb notebook**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Using this kind of logistic regression for NLP and before we got to that point, we were talking about using naive Bayes for NLP, and the basic idea was that we could take a document write. A review like this movie is good and turn it into a bag of words, representation consisting of the number of times each word appears right and we call this the vocabulary. This is the unique list of words. Okay and we used the SK, learn count vectorizer to automatically generate both the vocabulary which in SK low and they call they call the features and to court, create the bag of words, representation, z' and the whole group of them, then, is called a term document matrix. Okay and we kind of realized that we could calculate the probability that a positive review contains the word this by just averaging. The number of time disappears and the positive reviews, and we could do the same for the and we could do the same for the negatives. All right and then we could take the ratio of them to get something which, if it's greater than one was a word that appeared more often in the positive reviews or less than one was a word. That appeared more often than the negative reviews. Okay and then we realized, you know using using Bayes rule that we and taking the logs that we could basically end up with something where we could add up the logs of these plus the log of the ratio of the probabilities that things are in class.

One versus class zero and end up with something we can compare to zero. If it's be greater than zero, then we can predict a document is positive or if it's less than zero, we can predict the document is negative and that was our Bayes rule all right. So we kind of did that from math first principles and I think we agreed that the the naive in naive Bayes was a good description because it assumes independence when it's definitely not true. But it's an interesting starting point and I think it was interesting to observe when we actually got to the point where, like okay, now we've, you know calculated the the ratio of the probabilities and took the log and now, rather than multiply them together. Of course, we have to add them up and when, when we actually wrote that down, we realized like. Oh, that is, you know, just a standard weight, matrix product plus a bias right, and so then we kind of realized like oh okay, so like. If this is not very good accuracy, eighty percent accuracy, why not improve it by saying hey, we know other ways to calculate a cut. You know a bunch of coefficients and a bunch of biases, which is to learn them in a logistic regression. Alright, so in other words this this is the meanwhile, we use for a logistic regression, and so why don't we just create a logistic regression and fit it okay, and it's going to be give us the same thing, but rather than coefficients and biases, which are theoretically Correct based on you know this assumption of Independence and based on Bayes rule there'll be the coefficients and biases that are actually the best in this data right, so that was kind of where we got to, and so the kind of key insight here is like just About everything I find a machine learning ends up being either like a tree or you know a bunch of matrix products and monomi era.

T's right, like it's everything, seems to end up kind of coming down to the same thing, including, as it turns out Bayes rule. Okay and then it turns out that nearly all of the time, then, whatever the parameters are in that function, nearly all the time it turns out that they're, better learnt then calculated based on the theory right and indeed that's what happened when

we actually tried learning those Coefficients we got, you know 85 % okay. So then we noticed that we could also, rather than take the whole term document matrix. We could instead just take them the you know ones and zeros for presence or absence of a word, and you know. Sometimes it was, you know this equally as good, but then we actually tried something else, which is we tried, adding regularization and with regularization the binarized approach turned out to be a little better, all right, so then regularization was where we took the loss function and again, Let's start with our MSE and then we'll talk about cross-entropy loss function was our predictions, our actuals sum that up take the average plus a penalty. Okay, and so this specifically, is the l2 penalty. If this instead was the absolute value of W, then that would be the l1 penalty. Okay, we also noted that we don't really care about the loss function per se.

We only care about its derivatives, that's actually the thing that updates the weights, so we can, because this is a sum we can take the derivative of each part separately, and so the derivative of this part was just that right, and so we kind of learnt that Even though these are mathematically equivalent, they have different names. This version, it's called weight decay and it's kind of what's used. That term is used in the neural net.

3. 00:16:30

- **Cross-Entropy as a popular Loss Function for Classification (vs RMSE for Regression)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Literature, okay, so cross-entropy. On the other hand, you know it's just another. Loss function like root mean squared error, but it's specifically designed for classification, alright, and so here's an example of a binary cross-entropy. So let's say this is our you know. Is it a cat or a dog so as to say, is cat one or a zero, so Cat Cat Cat - and these are our predictions - this is the output of our final layer of our neural net or a logistic regression or whatever all right, then all we Do is we say: okay, let's take the actual times the log of the prediction, and then we add to that 1 minus actual times the log of 1 minus the prediction and then take the negative of that whole thing right. So I suggested to to you all that you tried to kind of write the if statement version of this, so hopefully you've done that by now. Otherwise, I'm about to spoil it for you. So this was Y times log y plus 1 minus y times. Log 1. Minus y right and negative of that, okay, so here wants to tell me how to write this as an if statement can she hit me okay, if y equal to sorry, if y equal to 1 mm-hmm, then in return I love Y mm-hmm, otherwise, well um else Return: log. 1, minus 1. Okay, oh that's the things at brackets and you take C. My name is good, so the key inside Chen she's using is that Y has two possibilities: 1 or 0. Okay, and so very often the math can hide the key insight which I think happens here until you actually think about what the values it can take right.

So that's that's all it's doing. It's saying either give me that or give me that all right could you pass that to the back? Please Jason, all right, I'm missing something, but do you know the two variables in that statement? If you got Y soon, it'd be like white hat on the wires yeah yeah. Thank you as usual. It's me missing something: okay, okay and so then the you know the multi category version is just the same thing, but you're saying you know it four different. More than just y equals one or zero, but y equals zero. One two three four: five: six, seven, eight nine, for instance, okay, and so that you know that loss function has a you, can figure it out yourself in particularly simple derivative, and it also you know. Another thing you could

play with at home. If you like is like thinking about how the derivative looks, when you add a sigmoid or a softmax before it, you know it turns out, it all turns out very nicely, because you've got an X P thing going into a log e thing. So you end up with you know very well behaved derivatives. The reason, I guess, there's lots of reasons that people use IR MSE for aggression and cross-entropy for classification, but most of it comes back to this statistical idea of a best, linear, unbiased estimator. You know and based on the likelihood function that kind of turns out that these have some nice statistical properties it turns out. However, in practice, root means great error.

In particular, the properties are perhaps more theoretical than actual, and actually nowadays using the the absolute deviation rather than some as grads deviation can often work better. So in practice, like everything in machine learning, I normally try both for particular data set I'll, try both loss functions and see which one works better and us, of course, it's a cattle competition, in which case you're told how capital is going to judge it, and you Should use the same loss function as caracals evaluation metric all right, so yeah, so this is really the key insight is like hey, let's, let's not use theory, but instead learn things from the data, and you know. We hope that we're going to get better results, particularly with regularization we do and then I think the key regularization insight here is

4. 00:21:30

- **Creating more NLP features with Ngrams (bigrams, trigrams)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Hey, let's not like try to reduce the number of parameters in our model, but instead like use lots of parameters and then use regularization to figure out which are actually useful. And so then we took that a step further by saying: hey, given we can do that with wrecker ization. Let's create lots more features by adding by grams and trigrams. You know, by grams, like by vast and by vengeance and trigrams, like by vengeance and by Vera, miles right, and you know just to keep things a little faster. We limited it to 800,000 features, but you know, even with the full 70 million features it works. Just as well and it's not a hell of a lot slower, so we created a term document matrix again using the full set of engrams for the training set, the validation set. And so now we can go ahead and say: okay, our labels as the training set labels as before our independent variables is the binarized term document matrix as before, and then let's fit a logistic regression to that and do some predictions and we get 90 % accuracy. So this is looking pretty good, okay, so the logistic regression. Let's go back to our naive Bayes right in our naive Bayes. We have this term document

5. 00:23:01

- **Going back to Naive Bayes and Logistic Regression,**
- **then 'We do something weird but actually not that weird' with "x_nb = x.multiply®"**
- **Note: watch the whole 15 mins segment for full understanding.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Matrix and then for every feature, we're calculating the probability of that feature occurring if it's class 1 that probability of that feature occurring if it's plus 2 and then the ratio of those two all right and in the paper that we're actually load basing this off. They call this P, this Q and this right, maybe I should just fill that in P here, maybe then we'll say probability to make it more obvious, okay and so then we kind of said: hey, let's, let's not use these ratios as the coefficients in that. In that matrix multiply, but let's instead like try and learn some coefficients. You know so maybe start out with some random numbers. You know and then try and use stochastic gradient descent to find slightly better ones. So you'll notice, you know some important features here. The R vector is a vector of Rank 1 and it's length is equal to the number of features and, of course, our logistic regression coefficient matrix is also of length. 1. Sorry Rank 1 and length 1/4. The number of features right - and we know we're saying like they're kind of two ways of calculating the same kind of thing right, one based on theory, one based on data. So here is like some of the numbers in R right. Remember it's using the log. So these numbers, which are less than zero, represent things which are more likely to be negative, and these ones are here are more likely.

So this one here is more likely to be positive, and so here's E ^ that - and so these are the ones we can compare to one rather than to zero. So I'm going to do something that hopefully, is going to seem weird and so first of all, I'm going to talk about. I got to say what we got to do and then I'm gon na try and describe why it's weird and then we'll talk about why it may not be as weird as we first thought. So here's what we got to do we're going to take our term document matrix and we're going to multiply it by. So what that means is we're gon na. I can do it here in Excel right, so we're going to say: let's grab everything in our term document matrix and multiply it by the equivalent value in the vector of our right. So this is like a broadcasted. Element-Wise multiplication, not a matrix multiplication. Okay and that's what that does okay, so here is the value of the term document matrix times R, in other words everywhere, that a zero appears there. A zero appears here and every time a one appears here, the equivalent value of R appears here. So we haven't really: we haven't really changed much all right, we've just we've just kind of changed the ones into something else than two into the into the odds from that feature, all right, and so what we're now going to do is we're going to use this As our independent variables, instead in our logistic regression, okay, so here we are multiply x, x, NB, x, naive, Bayes version is x times, R and now, let's do a logistic regression fitting using those independent variables.

And let's then do that for the validation set. Okay and get the predictions, and lo and behold we have a better number okay. So let me explain why this hopefully seem surprising, given that we're just multiplying. Oh, I picked out the wrong ones. I should have said ah not going. Okay, that's actually uh. I got the wrong number okay, so that's our independent variables right and then the the logistic regression has come up with some set of coefficients. Let's pretend for a moment that these are the coefficients that it happened to come up with. Okay, we could now say: well, let's not use this set, let's not use this set of independent variables, but let's use the original binarized feature matrix right and then divide all of our coefficients by the values in R and we're going to get exactly the same result. Mathematically, so you know, we've got our X naivebayes version of the independent variables and we've got some some set of weights, some some sort of coefficients I'll call it W right W one, let's see where it's found like this is a good set of coefficients for making Our predictions from right that X and B is simply equal to x times as in element wise x, ah right so in other words, this is equal to x, times ah times the weights and so like. We could just change the weights to be that right and get the same number. So this ought to mean that the change that we made to the dependent variable shouldn't have made any difference, because we can calculate exactly the

same thing without making that change.

So there's the question: why did it make a difference? So, in order to answer this question, I got to try and get you all to try and think about this. In order to answer this question, you need to think about like okay. What are the things that aren't mathematically the same? Why is why? Is it not identical? What are the reasons like come up with some hypotheses? What are some reasons that maybe we've actually ended up with a better answer and to figure that out we need to first of all start with like well. Why is it even a different answer? Why is that different? To that? Is it subtle all right? What do you think I just wondering if there was two different kinds of multiplications? You said that one is the element wise multiplication. No, they do end up mathematically being the same okay, pretty much, there's a minor in corporate. Not it's not that it's not some order! Operation, see, let's try kimchee, you are on a roll today. So let's see how you go, I feel, like Z features, aren't less correlated to each other. I mean I've made a claim that these are mathematically equivalent. So so what are you saying? Really? You know why are we getting different answers? It's good keep on coming up with hypotheses. We need lots of wrong answers before we start finding. It's really right ones. It's like that. You know warmer hotter colder. You know Ernest you're gon na get as hot.

Oh, does it have anything to do with the regularization? Ah, yes, and is it the fact that when you, let's start there right so Ernest point here is like okay, Jeremy you've said they're equivalent but they're equivalent outcomes right, but you got through and through a process to get there and that process included regularization and they're. Not necessarily equivalent regularization, like our loss function as a penalty, so yeah help us think through Ernest how much that might impact things well. This is I'm just noticing that the numbers are bigger in the ones that have been weighted by the naive phase: mm-hmm our weights, and so these are bigger and some are smaller. Some are bigger right, but there are some bigger ones, like the variance between the columns is much higher. The variance is bigger. Yeah. I think that's a very interesting insight. Okay, that's all yeah! Okay, so build on that prince has been on a roll or month. So here's not sure is it also consider like considering the dependency of different words instead, why this performing better, rather than all but independent of each other? No, really I mean it's it's you know again. Theoretically, these are creating mathematically, equivalent outputs, so they're not they're, not doing something different, except, as Ernest mentioned, they're getting impacted differently by regularization. So, what's so, what's regularization right regularization? Is we start out with our? That was the weirdest thing.

I forgot to go to screenwriting mode, and it just turns out that you can actually write in Excel and I had no idea that was true. I still use screenwriting Rosewood's could kill up my spreadsheet I'd, never trade, so our loss was equal to like our cross, entropy loss. You know based on the predictions of the predictions and the actuals right, plus our penalty. So, if your, if your weights a large right, then that piece gets bigger right and it drowns out that piece right, but that's actually the piece we care about right. We actually want it to be a good fit, so we want to have as little regularization going on as we can get away with. We want so we want to have less weights. So here's the thing right, our value. Yes, can you pass it over here? We should let less weights. Did you mean lesser weights? I do yeah yeah and I use the two words are level equivalently, which is not quite fair. I agree, but the idea is that weights that are pretty close to zero were kind of not there. So here's the thing, our values of ah you know and I'm not a Bayesian weenie, but I'm still gon na use the word prior right, they're kind of like a prior so like. We think that the the different levels of importance and positive or negative of these different features might be something like that right. We think that, like bad, you know might be more correlated with negative, then and good right. So our kind of

implicit assumption the before was that we have no priors.

So in other words, when we'd said squared weights, we're saying a nonzero weight is something we don't want to have right. But actually, I think what I really want to say is that differing from the naive Bayes expectation is something I don't want to do right, like only very from the naive Bayes prior, unless you have good reason to believe otherwise right, and so that's actually what this Ends up doing right, we end up saying you know what we think this value is probably three right, and so, if you're going to like make it a lot bigger or a lot smaller right, that's going to create the kind of variation in weights. That's going to cause that squared term to go up right. So so, if you can, you know just leave all these values about similar to where they are now all right, and so that's what the penalty term is now doing right. The penalty term, when our inputs is already multiplied by our is saying, penalize things where we're burying it from our naive Bayes prior. Can you pass it? Why multiply only with our not constant like a square or something later, when the variance would be much higher deciding because our our prior comes from an actual theoretical model right, so I said like I don't like to rely on the theory, but I have, if I Have some theory, then you know, maybe we should use that as our starting point rather than starting off by assuming everything's equal.

So our prior said: hey we've got this model coordinate phase and the naive Bayes model said if the naive Bayes assumptions were correct, then R is the correct coefficient right in this specific formulation that that's why we pick that because our our prior is based on that. That theory, okay, so this is a really interesting insight which I never really see covered, which is this idea is that we can use these. Like you know, traditional machine learning techniques we can imbue them with this kind of Bayesian sense by by starting out. You know incorporating our theoretical expectations into the data that we give our model right and when we do so that then means we don't have to regularize as much and that's good right, because if we regularize a lot, let's try it. Let's go back to you know. Here's our remember the the way they do it in the eschaton logistic regression is. This is the reciprocal of the amount of vectorization penalty, so will kind of add lots of regularization by making a small so that, like really hurts that really hurts our accuracy, because now it's trying really hard to get those weights down the loss function is overwhelmed by The need to reduce the weights and the need to make it predictive is kind of now seen as totally unimportant right. So so, by kind of starting out and saying you know what don't push the weights down so that you end up ignoring the the terms, but instead push them down so that you try to get rid of.

You know ignore differences from our expectation based on the naive Bayes formulation, so that ends up giving us a very nice result, which actually was originally this. This

6. 00:39:45

- **'Baselines and Bigrams: Simple, Good Sentiment and Topic Classification' paper by Sida Wang and Christopher Manning, Stanford U.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Technique was originally presented, I think about 2012 Chris Manning who's, a terrific NLP researcher, up at Stanford and CETA Wang, who, I don't know, but I assume, is awesome because his paper is awesome. They basically came up with this with this idea and what they did was they compared it to a number of other approaches on a number of other data sets. So

one of the things they tried is this one is the IMDB data set that and so here's naive, Bayes SVM on diagrams and, as you can see, this approach outperformed the other linear based approaches that they looked at and also some restricted, Boltzmann machine kind of Neural net based approaches they looked at now. Nowadays there are better ways there are. You know there are better ways to do this and in fact, in the deep learning course we showed new state-of-the-art result that we just developed at first, a a that gets well over. Ninety four percent - but still you know like particularly for a linear technique - that's easy, fast and intuitive. This is pretty good and you'll notice when they, when they did this, they only used by grams, and I assume that's because they I looked at their code and it was kind of pretty slow and ugly. You know I figured out a way to optimize it. A lot more as you saw, and so we were able to use here, try grams, and so we get quite a lot better. So we've got ninety one point: eight versus a ninety one point two, but other than that. It's identical.

Also, I mean they used a support, vector machine which is almost identical to a logistic regression in this case. So there's some minor differences right. So I think that's a pretty cool result and you know I will mention you know what you get to see here in class is the result of like many weeks and often many months of research that I do, and so I don't want you to think like This stuff is obvious: it's not at all like reading this paper, there's no description in the paper of like why they use this model, how it's different, why they thought it works. You know it took me a week or two to even realize that it's kind of like mathematically equivalent to a normal, logistic regression and then a few more weeks to realize that the difference is actually in the regularization. You know like this is kind of like machine learning, as I'm sure you've noticed from the Carroll competitions you enter. You know, like you, come up with a thousand good ideas: 999 of them, no matter how confident you are they're going to be great, they always turn out to be you know, and then, finally, after four weeks, one of them finally works and kind of gives you The enthusiasm to spend another four weeks of misery and frustration. This is the norm right and and like for sure that the the best practitioners I know in machine learning all share one particular trait in common, which they're very, very tenacious.

You know also known as stubborn and bloody-minded right, which is definitely a reputation. I seem to have probably fare along with another thing, which is that they're all very good coders. You know they're very good at turning their ideas into new code, so yeah. So you know this was like a really interesting experience for me working through this a few months ago to try and like figure out how to at least you know how to explain why this at there, at the time kind of state-of-the-art result, exists and so

7. 00:43:31

- **Improving it with PyTorch and GPU, with Fastai Naive Bayes or 'Fastai NBSVM++' and "class DotProdNB(nn.Module):"**
- **Note: this long section includes lots of mathematical demonstration and explanation.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Once I figured that out, I was actually able to build on top of it and make it quite a bit better and I'll. Show you what I did, and this is where it was very, very handy to have pytorch at my disposal, because I was able to kind of create something that was customized. Just the way

that I want it to be, and also very fast by using the GPU, so here's the kind of fastai version of the NBS vm. Actually, my friend Steven marady who's, a terrific researcher in NLP, has christened this. The NBS VM plus plus, which I thought was lovely, so here's that, even though there is no SVM, it's a logistic regression but, as I said, nearly exactly the same thing, so let me first of all show you like the code, so this is like we try To like once, I figure out like okay, this is like the best way I can come up with to do a linear bag of words model. I kind of embed it into fastai, so you can just write a couple of lines of code, so the code is basically hey. I want to create a data class for text classification. I want to create it from a bag of words. All right here is my bag of words. Here are my labels. Here is the same thing for the validation set and use up to 2,000 unique words per review right, which is plenty so then, from that model data construct, a learner which is kind of the faster. I generalization of a moral which is based on a dot product of naive, Bayes and then fit that model and then do a few epochs and after five epochs.

I was already up to ninety two point: two okay, so this is now like. You knowgetting quite well above this, this linear based one. So let me show you the code for for that, so the code is like horrifyingly short. That's it right and it'll also look on the whole extremely familiar right. There's, if there's a few tweaks here, pretend this thing that says embedding pretend it actually says: linear, okay, I'm going to show you embedding in a moment pretend it says linear. So we've got basically a linear layer where the number of features coming with the number of features as the rows and remember SK, learn, features, means number of words basically and then for each row. We're going to create one weight, which makes sense right for like a logistic regression, every every sort of for each row for each word, each word has one weight and then we're going to be multiplying it by the our values. So each word we have one our value per class, so I actually made this so this can handle like not just positive versus negative, but maybe figuring out, like which author created this work, they're, cooking, five or six authors, whatever right and basically we kind of use. Those linear layers to to get the the value of the weight and the value of the R, and then we take the weight times the R and then sum it up, and so that's just a dot product. Okay, so just just a simple dot product.

Just as we would do for any logistic regression and then do the softmax, so the very minor tweak the we add to get the the better result. Is this the main one really is this year, this plus something right and the thing I'm adding is it's a parameter, but I pretty much always use this. This version of this value 2.4. So what does this do so? What this is doing is it's again kind of changing the prior right. So, if you think about it, even once we used this R times the term document matrix as their independent variables, you really want to start with a question. Okay, the penalty terms are still pushing W down to 0 right. So what did it mean for W to be 0 all right? So what would it mean if we had? You know coefficient 0, 0, 0, 0, 0, all right. So what that would do when we go.okay this matrix times these coefficients, we still get 0 right, so a weight of 0 still ends up saying. I have no opinion on whether this thing is positive or negative. On the other hand, if they were all one right, then it's basically says my opinion is that the naive Bayes coefficients are exactly right. Okay, and so the idea is that I said 0 is almost certainly not the right prior right. We shouldn't really be saying: if there's no coefficient, it means ignore the naive Bayes coefficient. One is probably too high right, because we actually think that naive, Bayes is only kind of part of the answer.

All right - and so I played around with a few different data - sets where I basically said: take the weights and add to them some constant right and so 0 would become in this case 0.4. All right so, in other words, the the regularization penalty, is pushing the weights not towards zero, but towards this value right, and I found that across a number of data sets zero point.

Four works pretty well, but and it's pretty resilient alright. So again, this is the basic idea is to kind of like get the best of both worlds. You know where, where we're learning from the data using a simple model but we're incorporating you know our prior knowledge as best as we can, and so it turns out when you say: okay, let's, let's tell it, you know, as white matrix of zeros actually means that You should use about, you know about half of the values that ends up that ends up working better than the prior, that the weights should all be 0. Yes, is the the weights, the W? Is it that the point for the amount of regression required the amount of so we have this? You know bad things. We have the term where we reduce the amount of error. The prediction error - rmse plus we have the regularization - and is it W to the point for denote the amount of visualization required, so W are the weights right. So this is calculating our activations okay, so we calculate our activations as being equal to the weights times. There are some right, so that's just our normal, normal linear function right.

So so the the thing which is being penalized is my weight matrix. That's what gets penalized so by saying hey, you know what don't just use W use W plus 0.4. So that's not being penalized, it's not part of the weight matrix. Okay, so effectively the weight matrix gets point four for free yeah. So, by doing this even after regularization, then every I'm sorry every feature is getting some form of fate, some form of weight or something um, not necessarily because it could end up choosing a coefficient of negative 0.4 for a feature, and so that would say you know What even, though, though naive Bayes says it's the AH should be whatever for this feature, I think you should totally ignore it. Yeah great questions. Okay, we started at 20 past okay, let's take a break for about eight minutes or so and start back about 25. Okay, so a couple of questions at the break. The first was just for a kind of reminder or a bit of a summary as to what's going on here right and so here we have W plus I'm writing it out. Yeah plus adjusted weight of weight adjustment times right, so so normally what we were doing so normally what we are doing is saying: hey, logistic regression is basically WX right, I'm going to ignore the bias okay and then we were changing it to be W dot times X right and then we were kind of saying: let's do that bit first right, although in this particular case actually now I look at it, I'm doing it in this code.

It doesn't matter, obviously in this code, I'm actually doing and during this bit first, and so so this thing here, actually I call it W, which is probably pretty bad, it's actually W times X right. So so, instead of W times X times, R, I've got W times X, plus a constant times R right. So the key idea here is that regularization can't draw in yellow that's fair enough. Regularization wants the weights to be 0 right because we're trying it's trying to reduce that okay, and so what we're saying is like. Ok, we want to push the weights towards 0 because we're saying like that's our like default starting point expectation is the weights 0, and so we want to be in a situation where, if the weights are 0, then we have a model that like makes theoretical or Intuitive sense to us right this model if the weights are 0, doesn't make intuitive sense to us right, because it's saying hey multiply. Everything by 0 gets rid of all of that and gets rid of that as well, and we were actually saying no. We actually think our R is useful. We actually want to keep that right so so, instead we say you know what let's take that piece here and add 0.4 to it all right. So now, if the regularizer is pushing the weights towards 0, then it's pushing the value of this sum towards 0.4 right, and so therefore, it's pushing a whole model to 0.4 times R right so, in other words, our kind of default starting point.

If you've regularize to all the weights out altogether is to say yeah, you know, let's use a bit of our that's, probably a good idea: okay, . So that's the idea right! That's the idea is basically, you know what happens when when that's zero, but you and you want that to like be something sensible, because otherwise regularizing the weights to move in that direction,

wouldn't be such a good idea. Okay, second question was about engrams, so the N in Engram can be uni by try, whatever one two three, whatever grounds so so the this movie is good right. It has four unique grams. This movie is good. It has three by grams. This movie movie is, is good. It has two trigrams. This movie is movie is good, okay doc. Can you pass it? Do you mind, go back to the double ad change that zero point four stuff yeah. So I was wondering if this adjustment will harm the predictability of the model because think of extreme extreme case, if it's not zero point four, if it's four thousand and or black coefficients will be like right. So so exactly so, so our prior needs to make sense, and so our prior here - and you know this is why it's called dot prod NB is there prior is that this is something where we think naive. Bayes is a good prior right and so naive. Bayes says that our equals p over. That's not how you write P P over Q. I have not had much sleep P over Q is a good prayer, and not only do we think it's a good prior that we think our times X, plus B is a good model.

That's that's the naive Bayes model, so in other words, we expect that you know a coefficient of one is a good coefficient, not not 4,000 yeah. So we think specifically, we don't think we think 0 is probably not a good coefficient all right, but we also think that maybe the naive Bayes version is a little overconfident, so maybe one's a little high. So we're pretty sure that the right number, assuming that our moral only Bayes model is appropriate, is between 0 & amp 1. No. But what I was thinking is as long as it's not 0. You are pushing those coefficients that are supposed to be 0 to something, not zero and make the like high coefficients less distinctive from 0 coefficients well, but you see they're not supposed to be 0 they're supposed to be are like that's that's what they're supposed to be They're supposed to be are right and so and remember this is inside our forward function, so this is part of what we're taking the gradient of right. So it's basically saying: okay, we're still gon na you know you can still set self W to anything you like, but just the regularizer wants it to be zero, and so all we're saying is okay. If, if you want it to be zero, then I'll try to make 0 B, you know give a sensible answer. That's the basic idea and, like yeah, nothing says, point fours perfect.

For every data set, I've tried a few different data sets and found various numbers between point three and point six that are optimal, but I've never found one. Where point four is less good than zero, which is not surprising, and I've also never found one where one. It's better right, so the idea is like this is a reasonable default, but it's another parameter you can play with which I kind of like right. It's another thing you could use grid, search or whatever to figure out from your data set what's best, and you know really the key here being every model before this one, as far as I know, has implicitly assumed it should be zero because they just they don't Have this parameter right and you know by the way I've actually got a second parameter here as well, which is the same thing I do to R, is actually divided by a parameter which I'm not going to worry too much about it. Now but again it's another parameter. You can use to kind of adjust what the nature of the regularization is. You know, and I mean in the end I'm a empiricist, not a theoretician. You know the. I thought this seemed like a good idea. Nearly all of my things that seem like a good idea, turn out to be stupid. This particular one Dave good results. You know on this data set and a few other ones as well.

Okay, could you pass that newest data yep yeah, I'm sure a little bit confused about the W plus W? Is it huh, so you mentioned that we do W plus W adjusted so that the coefficients don't get set to zero, that we place some importance on the priors. But you also said that the the effect of learning can be that wget set to a negative value which is mentally W, plus W right zero. So if, if we are, we are allowing the learning process to indeed set the priors to zero. So why is that in any way different from just having W because yeah great question, because of regularization, because we're penalizing it by that right? So, in other words, we're

saying you know what, if you're, if the best thing to do is to ignore the value of R that'll cost you you're going to have to set W to a negative number right. So only do that, if that's fairly a good idea. Unless it's clearly a good idea, then you should leave, leave it where it is that that's the only reason like all of this stuff we've done today is basically entirely about. You know maximizing the advantage we get from regularization and saying regularization pushes us towards some default assumption and nearly all of the machine learning literature assumes that default assumption is everything zero and I'm saying like it turns out. You know it makes sense theoretically and turns out empirically that, actually you should decide what your default assumption is and that'll give you better results.

So would it be right to say that, in a way you are putting an additional hurdle in the along the way towards getting all coefficients to zero, so it will be able to do that if it is really worth it yeah. Exactly so I'd say like the default herd or without this is, is making a coefficient non. Zero is the heck hurdle, and now I'm saying no, the coefficient not be equal to 0.40. So this is some of the W squared in to see some of into some lambda or C penalty. Constant yeah yeah times something yeah, so the beta K should also depend on the value of C. If it is very less like. If C is, I say to you, hey yeah, so if a is point one, then the wage might not go towards zero yeah. Then we might not need weight decay so well that the whatever this value I mean, if the if the value of this is zero, then there is no recordation right, but if this value is higher than zero, then there is some penalty right and - and presumably we've Set it to non zero because we're overfitting, so he wants some penalty, and so if there is some penalty, then then my assertion is that we should penalize things that are different to our prior, not that we should penalize things that are different to zero and prior Is that things should be, you know around about equal to our ok, let's move on thanks for the great questions I want to talk about. Embedding I said pretend it's linear, and indeed we can pretend it's linear.

Let me show you how much we can pretend it's! Linear, as in n n dot, linear create a linear layer here is our data matrix. Alright, here are our coefficients. If we're in the R version here are coefficients are right. So if we were to put those into a column vector like so right, then we could do a matrix multiply of that by that right and so we're going to end up with so here's our matrix, here's our vector right so we're going to end up with 1 times 1 plus 1 times 1, 1 times 1, 1 times 3 right, 0 times, 1, 0 times, point 3, all right and then the next one, 0 times 1, 1 times 1, so forth. Ok, so like that, the matrix multiply you know of this independent variable matrix by this coefficient matrix is going to give us an answer. Ok, so that's that is just a matrix multiply. So the question is like ok. Well, why didn't Jeremy right and n minion? Why did Jeremy right and n dot embedding and the reason is because if you recall, we don't actually store it like this, because this actually of whit's 800,000 and of height 25,000 right. So, rather than storing it like this, we actually store it as 0. 1. 2. 3 right, 1, 2. 3. 4. 0. 1. 2. 5. 1. 2, 4, 5. Okay, that's actually how we store it, that is this bag of words contains, which word indexes. That makes sense. Ok, so that's like this is like a sparse way of storing it right.

It's just list out the indexes in each sentence, so, given that I want to now do that matrix multiply that I just showed you to create that same outcome right, but I want to do it from this representation. So if you think about it, all this is actually doing. Is it saying a 1 hot? You know this is basically one hot encoded right. It's kind of like a dummy dummy matrix version. Does it have the word this doesn't have? The word movie doesn't have the word is and so forth. So if we took the simple version of like doesn't have the word this 100 right and we multiplied that by that right. Then that's just going to return the first item. That makes sense so in general, a one hot encoded vector times a matrix, is identical to looking up that matrix to find the end row in it all right. So this is identical to saying, find the zero first,

second and fifth coefficients right, so they're they're the same they're exactly the same thing and like it doesn't like in this case, I only have one coefficient per feature right, but actually the way I did this was To have one coefficient per feature for each class right, so in this case is both positive and negative, so I actually had kind of like an AA, positive and a negative, so our negative would be just the opposite right equals that divided by that now, in the Binary case, obviously it's redundant to have both, but what if it was like? What's the author of this text? Is it Jeremy or Savannah or Terrence right now, we've got three categories. We want three values of R right, so the nice thing is doing this sparse version.

You know you can just look up. You know the 0th and the first and the second and the fifth. Alright and again it's identical mathematically, identical to a multiplying by a one Haughton coded matrix. But when you have sparse inputs, it's obviously much much more efficient. So this computational trick, which is mathematically identical to not conceptually analogous to mathematically identical to multiplying by a one hot encoded matrix, is called an embedding right. So I'm sure you've all heard - or most of you probably heard about embeddings, like word embeddings word to their core glove or whatever and people love to make them sound like this amazing, new, complex, neural net thing right, they're not embedding means make a multiplication by a One hot encoded matrix faster by replacing it with a simple array: cup. Okay, so that's why I said you can think of this, as if it said self, W equals n n dot, linear and F, plus one by one right, because it actually does the same thing right. It actually is a matrix with those dimensions. This actually is a matrix with those dimensions right. It's a linear layer, but it's expecting that the input we're going to give it is not actually one hot encoded matrix, but is actually a list of integers right, the indexes for each word of each item. So you can see that the forward function in fastai automatically gets for this Werner for feature indexes right, so they come from the sparse matrix automatically numpy makes it very easy to just grab those those indexes. Okay, so in other words there we've got here.

We've got a list of H, word index of a of the 800-thousand that are in this document, and so then this here says look up each of those in our embedding matrix, which is got 800,000 rows and return. Each thing that you find okay, so mathematically, identical to multiplying by the one hunting coded matrix so make sense. So that's all an embedding is, and so what that means is we can now handle building any kind of model. Like a you know, whatever kind of neural network, where we have potentially very high cardinality categorical variables as our inputs, we can then just turn them into a numeric code between zero and the number of levels. And then we can learn a you know, a linear layer from that, as if we had one hot encoded it without ever actually constructing the one hot encoded version and without ever actually doing that matrix model play okay. Instead, we will just store the index version and simply do the array lookup, okay and so the gradients that are flowing back. You know, basically, in the one hot encoded version, everything that was a zero has no gradient, so the gradients flowing back is best go to update the particular row of the embedding matrix that we used okay, and so that's fundamentally important for NLP. Just like here. Like you know, I wanted to create a pytorch model that would implement this. This ridiculously simple little equation, alright, and to do what, without this trick, would have meant.

I was feeding in a twenty five thousand by adherence to 800,000 element array, which would have been kind of crazy right, and so this this trick allowed me to write. You know you know: I've just replaced the word linear with embedding replace the thing that feeds the one-hot encodings in with something to dispense the indexes in, and that was it that then it kept working, and so this now trains you know in about a minute per Epoch, okay, so what we can now do is we can now take this idea and apply it not just to language but to anything

right, for example, predicting the sales of items at a grocery. Yes, where's that asset, just a quick question, so you're not actually looking up anything right. We are just seeing that now that array with the indices, that is the representation so the represent, so we are doing a lookup right. The representation that's being stored it for the book, but for the bag of words is now not one one 100 one, but oh one, two five right, and so then we actually have to do our matrix product right, but rather than doing the matrix product, we look Up the zero thing, and the first thing, and the second thing and the fifth thing so that means we are still retaining the one hard encoded matrix. No, we didn't there's no one hot encoded matrix used here this here's, the one who decoded matrix, which is not currently highlighted. We've currently highlighted the list of indexes and the list of coefficients from the weight matrix.

So it says: okay, okay, so what we're going to do now is we're kind of go to just go to go a step further and saying, like let's not use a linear model at all. Let's use a multi-layer neural network right and let's have the input to that potentially be include some categorical variables and those categorical variables we will just have as numeric indexes and so the first layer for those won't be a normal linear layer. There'll be an embedding layer which we know behaves exactly like a linear layer.

8. 01:17:30

- **Deep Learning: Structured and Time-Series data with Rossmann Kaggle competition, with the 3rd winning solution 'Entity Embeddings of Categorical Variables' by Guo/Berkhahn.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Mathematically and so then our hope will be that we can now use this to create a neural network for any kind of data. All right, and so there was a competition on Kaggle a few years ago - called rossmann, which is a German grocery chain, where they asked to predict the sales of items in in their stores right, and that included the mixture of categorical and continuous variables. And in this paper by Gordon Burke and they described their third-place winning entry, which was much simpler than the first placed winning entry, but nearly as good but much much simpler because they took advantage of this idea of what they call ng embeddings in the paper. They they thought, I think that they had invented this actually had been written before earlier by yoshua, bengio and his co-authors in another cackle competition, which was predicting taxi destinations. Although I will say, I feel, like war went a lot further in describing how this can be used in many other ways, and so will will talk about that as well. So the so, this one is actually in the is in the deep learning one repo. Okay deal one lesson: three: okay, because we talked about some of the deep learning specific aspects in the deep learning course, where else in this course we're going to be talking mainly about the feature, engineering and we're also going to be talking about. You know kind of this, this embedding idea. So let's start with the data right.

So the data was, you, know, store number one on the 31st of July 2015 was open, they had a promotion going on, there was a school holiday, it was not a state holiday and they sold five thousand two hundred and sixty three items. So that's the key data they provided, and so the goal is obviously to predict sales in a test set that has the same information without sales. They also tell you that for each store it's of some particular type, it sells some particular assortment of goods. Its nearest competitor competitor is some distance away. The competitor

opened in September 2008 and there's some more information about promos. I don't know the details of what that means like in many Carroll competitions. They let you download external data sets if you wish, as long as you share them with other competitors, so people - oh, they also told you what's date. Each store is, in so people downloaded a list of the names of the different states of Germany. They downloaded a file for each state in Germany for each week some kind of Google trend data. I don't know what specific Google trend they got, but there was that for each date they downloaded a whole bunch of temperature information, , that's it and then here's the test set. Okay, so I mean one interesting insight here. Is that the it was probably a mistake in some ways for Russman to design this competition as being one where you could use external data, because in reality you don't actually get to find out next week's weather or next week's Google Trends? You know, but you know, when you're competing in category you don't care about that.

You just want to win, so you use whatever you can get. So, let's talk first of all about

9. 01:21:30

- **Rossmann Kaggle: data cleaning & feature engineering.**
- **Using Pandas to join tables with 'Left join'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Data cleaning, you know that there wasn't really much feature engineering done in this third place, winning entry like by particularly by cattle standards, where normally every last thing counts. This is a great example of how far you can get with with a neural net, and it certainly reminds me of the claims prediction competition we talked about yesterday, where the winner did no feature engineering and entirely relied on deep learning, the laughter in the room. I guess is from people who did a little bit more than no feature engineering in that competition. So you know I should mention by the way, like I find that bit where, like you, work hard at a competition and then it closes and you didn't win and the winner comes out and says this is how I won like that's the bit where you learn. The most right, like sometimes that's, happened to me and it's been like. Oh I thought of that. I thought I tried that and then I go back and I realize I like had a bug there. I didn't test properly and I learn like oh okay, like I really need to learn to like test this thing in this away. Sometimes it's like. Oh, I thought of that, but I assumed it wouldn't work. I've really got to remember to check everything before I make any assumptions, and you know sometimes it's just like. Oh I I did not think of that technique.

Wow now I know it's better than everything I just tried because, like otherwise somebody says like hey, you know: here's a really good technique, you're like okay, great right, but when you spent months trying to do something and like somebody else, did it better by using that Technique, that's pretty convincing right and so like it's kind of hard, like I'm standing up in front of you saying here's a bunch of techniques that I've I've used and I've won some capital competitions and I've got some state of the art results. But it's like that's kind of second-hand information by the time it hits you right, so it's really great to yeah, try things out and and also like it's been kind of nice to see, particularly I've noticed in the deep learning course. Quite a few of my students have you know. I've said like this technique works really well and they've tried it and they've got into the top ten of a Keagle competition the next day and they're like okay, that that counts is working really well, so so yeah caryl competitions are helpful for lots and lots of Reasons, but you know one of the best ways is what happens after

it finishes, and so definitely like for the ones that you that are now finishing up, make sure you, you know, watch the forums, see what people are sharing in terms of their solutions, and you Know if you want to learn more about them like don't feel free to ask the winners like hey.

Could you tell me more about this or that people are normally pretty pretty good about explaining and then ideally try and replicate it yourself right and that can turn into a great blog post? You know, or a great kernel is to be able to say: okay such-and-such said that they use this technique. Here's a really short explanation of what that technique is, and here's a little bit of code showing how it's implemented, and you know, here's the results showing you. You can get the same result that can be a really interesting write-up as well. Okay, so you know it's, it's always nice to kind of have your data reflect like. Oh no bee is kind of easy to understand as possible. So in this case the data that came from Kegel used various you know, integers for the holidays. We can just use a boolean if like was it a holiday or not so like just clean that up? We've got quite a few different tables. We need to join them all together right. I have a standard way of joining things together with pandas. I just used the pandas merge function, and specifically, I always do a left joint. So, who wants to tell me what a left join is since it's there go ahead? You retain all the rows in the left table and you take so you have a key column. You match that with a key column in the right side table and you just merge the rules that are also present in the right side table yeah. That's a great explanation, good job. I don't have much to add to that.

The key reason that I always do a left join is that after I do the join, I always then check if there were things in the right-hand side that a noun no right, because if so it means that I some things yeah, I haven't shown it here, But I also check that the number of rows hasn't varied before and after, if it has, that means that the right hand side table wasn't unique. Okay, so even when I'm sure something's true, I always also assume that I've screwed it up. So I always check so I could go ahead and merge the state names into the whether I can also, if you look at the Google Trends table, it's got this week range which I need to turn into a date in order to join it right, and so The nice thing about doing this in pandas is that pandas gives us access to. You know all of Python right and so, for example, inside the the series object, Str attribute that gives you access to all the string. Processing functions not just like cat gives you access to the categorical functions. Dt gives you access to the date/time functions, so I can now split everything in that column and it's really important to try and use these pandas functions because they, you know they're going to be vectorized accelerated through you know, often threesome D, at least through you know, C code, so that runs nice and quickly, and then you know, as per usual, let's add date metadata to our dates. In the end, we are basically denormalizing all these tables we're going to put them all into one table.

So in the Google trend table there was also, though they were mainly trends by state, but there was also trends for the whole of Germany, so we kind of put the Germany on you know the whole of Germany ones into a separate data frame so that we Can join that so we're going to have that Google trend for this state and Google trend for the whole of Germany, and so now we can go ahead and start joining both for the training set and for the test set and then which both check that we Don't have zeros my merge function. I set the suffix. If there are two columns are the same. I set their suffix on the left to be nothing at all, so it doesn't screw around with the name and the right hand side to be underscore Y, and in this case I didn't want any of that. You look at ones, so I just went through and deleted them. Okay and then we're gon na in a moment we're going to try to create a competition. You know the the the main competitor for this store has been open since some date right, and so you can just use

pandas to date. I'm passing in the year the month and the day right, and so that's going to give us an error unless they all have years and months so so we're going to fill in the missing ones with the 1900 and a1 okay and then what we really know. It we didn't want to know is like how long is this store been open for at the time of this particular record all right, so we can just do a date.

Subtract! Okay! Now, if you think about it, sometimes the competition you know open later than this particular row, so sometimes it's going to be negative and it doesn't probably make sense to have negative spending like it's going to open in X days time now. Having said that, I would never put in something like this without first of all running a model with it in and without it in right, because, like our assumptions about about the data very often turned out not to be true. Now, in this case, I didn't invent any of these pre-processing steps. I wrote all the code, but it's all based on the third-place winners, github repo right, so knowing what it takes to get third place in the cable competition, I'm pretty sure they would have checked every one of these pre-processing steps and made sure it actually improved their Their validation set score okay, so what we're going to be doing is creating a neural network where some of the inputs to it are continuous and some of them are categorical, and so what that means in the in the neural net that you know we have we're. Basically going to have you know this kind of initial weight matrix right and we're going to have this. This input feature vector right, and so some of the inputs are just going to be plain. Continuous numbers, like you know, what's the maximum temperature here or what's the number of kilometers to the nearest store, and some of them are going to be one HUD encoded effectively right, but we're not actually going to store it as one hot encoded.

We're actually going to store it as the index right, and so the neural net model is going to need to know which of these columns should you should you basically create an embedding, for which ones should you treat you know as if they were kind of one Hot encoded, and which ones should you just you feed directly into the linear layer right and so we're going to tell the model when we get there, which is which, but we actually need to think ahead of time about, like which ones do we want to treat? As categorical, and which ones are continuous, in particular things that were going to treat it as categorical, we don't want to create more categories than we need right, and so let me show you what I mean. The the the third-place getters in this competition decided that the number of months that the competition was open was something that they were going to use as a categorical variable, all right, and so in order to avoid having more categories than they needed. They truncated it at 24 months. They said anything more than 24 months old truncate to 24. So here are the unique values of competition months open and it's all the numbers from naught to 24 right. So what that means is that there's going to be? You know an embedding matrix, that's going to have basically an embedding vector for things that aren't open yet for things that are open a month for things that are open two months and so forth. Now they absolutely could have done that as a continuous variable right. They could have just had a number here, which is just a single number of how many months has it been open and they could have treated it as continuous and fed it straight into the initial weight matrix.

What I found, though, and obviously what these competitors found is, where possible it's best to treat things as categorical variables right and the reason for that is that, like when you feed some things through an embedding matrix, you basically mean it means every level can be treated Like totally differently right, and so for example, in this case, whether something's been open for zero months or one months, is right, really different right. And so, if you fed that in as a continuous variable, it would be kind of difficult for the neural net to try and find

a functional form. That kind of has that that big difference as possible, because neural Nets can do anything right, but you're not making it easy for it. Where else, if you used an embedding treated it as categorical, then it will have a totally different vector for zero versus one right. So it seems like particularly as long as you've got enough data that the treating columns as categorical variables, where possible, is a better idea, and so I say when I say we're possible that kind of basically means like where the cardinalities not too high. You know. So if this was like, you know the sales ID number that was like uniquely different on every row, you can't treat that as a categorical variable right, because you know it would be a huge embedding matrix and everything only appears once or ditto for, like kilometers away From the nearest store to two decimal places, you wouldn't make a categorical variable all right, so that's kind of that's kind of the rule of thumb that they both used in this competition.

In fact, if we scroll down to their choices here is how they did it right, they're continuous variables, with things that were genuinely continuous, like number of kilometers away to the competitor, the temperature stuff right, the number you know the specific number in the Google trend right, Where else everything else, basically, they treated as categorical. Okay, so that's it for today, so yeah next time, we'll we'll finish this off we'll see we'll see how to turn this into a neural network and yeah kind of wrap things up so see. Then

Lesson 12

Outline

- NLP : trigrams
- Naive Bayes Classifier
- Binarized version of NB
- NBSVM - combination of probs
- Storage efficiency of 1-hot
- RossMan store examination
- Introduction to embeddings

Video Timelines and Transcript

Note: you may want to pay specific attention to the second part of this final lesson, where Jeremy brings up delicate issues on Data Science & Ethics.
This goes beyond what most courses on DS cover.

1. 00:00:01

- **Final lesson program !**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

I thought what we might do today is to like finish off where we were in this Russman noteblock, looking at timeseriesforecasting and structured data analysis, and then we might do a little mini review of like everything, we've learned because believe it or not. This is the end. Like there's nothing more to know about machine learning, rather than everything that you're going to learn next semester and for the rest of your life there anyway, I got nothing else to teach ya, so I'll, do a little review and and then we'll cover like the most Important part of the course which is like thinking about like what are the what what of how are ways to think about how to use this kind of technology appropriately, and you know effectively in a way that's a positive, hopefully a positive impact on society. So last time

2. 00:01:01

- **Review of Rossmann Kaggle competition with 'lesson3-rossman.ipynb'**
- **Using "df.apply(lambda x:…)" and "create_promo2since(x)"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

We got to the point where we talked a bit about this site. This idea that when we were looking at like building this competition months, open derived variable, but we actually truncated it

down to be no more than 24 months, and we talked about the reason. Why being that, we actually wanted to use it as a categorical variable, because categorical variables, thanks to embeddings, have more flexibility in how the neural net can can use them, and so that was kind of where we, where we left off. So, let's like keep working through this because what's happening in this notebook is stuff which is probably going to apply to most time series data sets that you work with right and, as we talked about like, although we used the F dot apply here. This is something where it's running a piece of Python code over every row and it's that's terrifically slow right, so we only do that if we can't find a vectorized, pandas or numpy function, that can do it too, the whole column at once, but in this case I couldn't find a way to convert a year and a week number into a date without using arbitrary Python. Also worth remembering. This idea of a lambda function, anytime, you're, trying to apply a function to every row of something or every element of a tensor, or something like that. If there isn't a vectorized version, already you're going to have to call something like data frame, dot apply which will run a function, you pass to every element, so this is something like you know.

Kind of this is basically a map in functional programming. Since very often, the function that you want to pass to, it is something you're just going to use once and then throw it away. It's really common to use this lambda approach, so this lambda is creating a function just for the purpose of telling DF not apply. What to use right so we could. We could also have written this in a different way, which would have been to say define create from o2 since on some value return, and then we put that in here, okay, so that - and that are the same thing. Okay, so one approach is to define the function and then pass it by name or the other is to define the function in place using lambda all right, and so, if you're not comfortable, creating and using lambdas. You know good thing to practice and playing around at the F dot, apply as a good way to good way to practice it. Okay,

3. 00:04:30

- **Durations function “get_elapsed(fld, pre):” using “zip()”**
- **Check the notebook for detailed explanations.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So let's talk about this durations section, which may at first seem a little specific, but actually it turns out not to be. What we're going to do is we're going to look at three fields: promot state holiday and school holiday, and so basically, what we have is a table of for each store for each date. That's that store have a promo going on at that date. Is there a school holiday in that region of that store of that date? Is there a state holiday in that region for that store at that date? Okay, and so this kind of thing is, you know, like their events and time. Series with events are like very, very common, like if you're looking at oil and gas drilling data, you're trying to say like the flow through this pipe. You know here's an event representing when it set off some alarm. You know or here's an event where the drill got stuck or or whatever right, and so like most time, series at some level will tend to represent some events. So the fact that an event happened at a time is is interesting itself, but very often a time series will also show some something happening before and after the event. So, for example, in this case we're doing grocery sales prediction if there's a holiday coming up. It's quite likely that sales will be higher before and after the holiday and lower during the holiday.

If this is a City based store right, because you know you're gon na you've got to stock up

before you go away to bring things with you, then, when you come back, you've got a refill. The fridge, for instance right. So it's alone like we, don't necessarily have to do this kind of feature engineering to create features specifically about like this is before or after a holiday that the neural net, you know the more. We can give the neuron that, like the kind of information it needs the less it's going to have to learn it, the less that it's going to have to learn it, the more we can do with the data, or we already have the more we can do With the you know, the size architecture we already have so future engineering, even even with stuff like neural nets, is still important because it means that you know we'll be able to do. You know, get better results with whatever limited data we have whatever limited computation. We have so the basic idea here, therefore, is when we have events in our time series is: we want to create two new columns for each event? How long is it going to be until the next time this event happens, and how long has it been since the last time that event happened so, in other words, how long until the next state holiday, how long since the previous state holiday? Okay, so that's not something which I'm aware of as existing as a library or anything like that, so we, I wrote up here by hand right and so importantly, I need to do this by store right. So I want to say like because you know for this store: when was this stores last promo? So how long has it been since the last time it had a promo how long it will be until the next time it has a promo, for instance, all right. So here's what I'm going to do! I'm gon na create a little function, that's going to take a field name and I'm going to pass it a chive, promo and then state holiday and then school holiday, all right! So let's do school holiday, for example.

So we'll say, field equals school holiday and then we'll say, get elapsed, school holiday, comma after so let me show you what that's going to do so. We've got a first of all sort by store and date right. So now, when we loop through this, we're going to be looping through within a store so store number one January, the first journey, the second January third and so forth, and as we look through each store, we're basically going to say like is: is this row a School holiday or not, and if it is a school holiday, then we'll keep track of this variable called last date, which says this is the last date and which, where we saw a school holiday, okay and so then we're basically going to append to our result. The number of days since the last school holiday - that's the kind of basic idea here. So there's a few interesting features. One is the use of zip right, so I could actually write this much more simply right, I could say: let's go through well, we could basically go through like four row in DF, get a rose right and then grab the the fields we want from each row. It turns out this is 300 times slower than the version that I have, and basically like iterating, through a data frame and extracting specific fields out of a row has a lot of overhead. What's much faster is to iterate through a numpy array. So if you take a series like the F store and add values after it, that grabs a numpy array of that series. Okay, so here are three numpy arrays one is the store.

Ids one is whatever field is, in this case: that's a school holiday and what is the date so now. What I want to want to do is look through the first one of each of those lists and then the second one of each of those lists and then the third one of each of those lists and like this is a really really common pattern. I need to do something like this in basically every notebook I write and the way to do it is with zip all right. So zip means look through each of these lists one at a time, and then this here is where we can grab that element out of the first list, the second list and the third list. Okay, so if you haven't played around watch with zip, that's a really important function to practice with, like I say, I use it in pretty much every notebook. I write all the time you have to look through. You know a bunch of lists at the same time. Alright, so we're going to look through every store every school holiday at every date. Yes, so is it living through, like all the possible combinations of each of those or for one one, one yeah exactly thanks for the question, so in this case we basically want to say: let's grab the

first store, the first school holiday, the first date right so Fast or one January, the first school holiday was true or false right, and so, if the, if it is a school holiday, I'll keep track of that fact.

By saying the last time I saw a school holiday was that day, okay and then append? How long has it been since the last school holiday right and if the store ID is different to the last door ID I saw, then I've now got to a whole new store, in which case I have to basically reset everything. Okay, you pass that to her. What will happen to the first points that we don't have a life last holiday yeah? So I just said I basically set this to some arbitrary starting point. It's going to end up with, like I can't remember, is either largest or the smallest possible date, and you know you may need to replace this with a missing value afterwards or some you know the zero or or whatever you know. The nice thing is, though, thanks to rallies, it's very easy for a neural net to kind of cut off extreme values. So in this case I didn't do anything special with it. I ended up with these like negative, a billion day time stamps, and it still worked. Fine, okay, so we can go through, and so the next thing to note is there's a whole bunch of stuff that I need to do to both the training set and the test set right. So in the previous section I actually kind of added this little loop, where I go for each of the training data frame and the test data frame through these things right. So I kind of you know each cell I did for each of the data frames. I've now got a whole coming up a whole series of cells that I want to run, first of all for the training set and then for the test set.

So in this case, the way I did that was, I have two different cells here: one which set DF to be the training set, one which set to be the test set, and so the way I use this is, I run just this cell right and then I run all the cells underneath right, so it does it alter the training set and then I come back and run just this cell and then run all the cells underneath. Okay, so like this, notebook is not designed to be just run from top to bottom, but it's designed to be run in this particular way, and I mentioned that because like this can be a handy trick. No, like you could, of course, put all the stuff underneath in a function that you pass the data frame to and call it once with a test set once with the training set bill, I kind of like to experiment a bit more interactively. Look at each step as I go, so this way is an easy way to kind of run something on two different data frames without turning it into a function. Okay, so this is going to. If, if I sort by store - and by date, then this is keeping track of the last time, something happened, and so this is therefore going to end up telling me how many days was it since the last school holiday? Okay. So now, if I sort date descending and call the exact same function, then it's going to say how long until the next school holiday, okay, so that's a kind of a nice little trick for adding these kind of event time as arbitrary event. Timers into your time, series models right.

So if you're doing, for example, the Ecuadorean groceries competition right now, you know, maybe this kind of approach would be useful for various events in that as well. Do it for state holiday, do it for promo there we go! Okay, the next thing

4. 00:16:10

- **Rolling function (or windowing function) for moving-average**
- **Hint: learn the Pandas API for Time-Series, it's extremely diverse and powerful**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

That we look at here is rolling functions, so rolling functions is how we rolling in pandas is how we, what we call create, what we call windowing functions. So let's say I had some data. You know something like this right and this is like date, and I don't know this is like sales or whatever. What I could do is, I could say, like: okay, let's create a window around this point of like seven days, all right, so it'd be like okay. This is a seven day window, say right, and so then I could take the average sales in that seven-day window. All right, then, I could like do the same thing like I don't know over here. Take the average sales over that seven-day window all right, and so, if we do that for every point and join up those averages you're going to end up with a moving average okay. So the kind of the the the more generic version of the moving average is a window function, ie, something where you apply to some function to some window of data around each point. Now very often that windows that I've shown here are not actually what you want. If you're trying to build a predictive model, you can't include the future as part of a moving average all right. So quite often you actually need a window. That's ends here, so that would be our window function right and so pandas lets you create window. Func arbitrary window functions using this rolling here. This here says how many time steps do I want to apply the function to write.

This here says if I'm at the edge, so in other words, if I'm like out here, should you have? Should you make that a missing value, because I don't have seven days to average over or you know, what's the minimum number of time periods to use? That's so here I said one okay and then then optionally, you can also say: do you want to say the window at the start of a period or the end of a period or the middle of the period? Okay, so, and then within that you can then apply whatever function. You like, okay, so here I've got, my weekly buy, store, sums; okay, so there's a nice easy way of getting kind of moving averages or or whatever else - and you know I should mention in pandas. If you go to the time series page on pandas, there's literally like look at just the index here time, series functionality is all of this. Is this like there's lots because, like where's bikini, who created this, he was originally in hedge fund trading? I believe - and you know his work was all about time series and so I think, like pandas originally was very focused on time series and still you know it's perhaps the strongest part of pandas. So, if you're playing like if you're playing around with time-series computations, you definitely owe it to yourself to try to learn this entire API and, like it, there's a lot of kind of conceptual pieces around like time, stamps and date, offsets and resampling and stuff like that.

To kind of get your head around, but it's totally worth it because otherwise you'll be writing this stuff as loops by hand it's going to take you a lot longer than leveraging what pandas already does and of course pandas will do it in you know highly optimized C code for you vectorize the C code, whereas your version is going to loop in Python. So definitely worth you know. If you're doing stuff in x here is learning the the full pandas time series API there's about is about as strong as any time series API out there. Okay, so at the end of all that, you can see here's those kind of starting point values. I mentioned slightly on the extreme side, and so you can see here the 17th of September store. One was thirteen days after the last school holiday. The 16th was 12 11 10, so forth, okay and we're currently in a promotion right here. This is one day before a promotion. Here, we've got nine days after the last promotion and so forth. Okay, so that's how we can add kind of event counters to our time series and probably always a

5. 00:21:40

- **Create Features, assign to ‘cat_vars’ and ‘contin_vars’**

- **'joined_samp', 'do_scale=True', 'mapper',**
- **'yl = np.log(y)' for RMSPE (Root Mean Squared Percent Error)**
- **Selecting a most recent Validation set in Time-Series, if possible of the exact same length as Test set.**
- **Then dropping the Validation set with 'val_idx = [0]' for final training of the model.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Good idea, when you're doing work with time series so now that we've done that you know, we've got lots of columns in our data set, and so we split them out into categorical versus continuous columns. We'll talk more about that in a moment in the review section, but so these are going to be all the things I'm going to create an embedding for okay, and these are all of the things that I'm going to feed directly into the into the model. So, for example, we've got like competition distance, so that's distance to the nearest competitor, maximum temperature, and here we've got day of week right so so here we've got maximum temperature. Maybe is like twenty two point: one because they use centigrade in Germany. We've got distance. Two nearest competitor might be 321 kilometres 0.7 all right and then we've got day of week, which might be I don't know. Maybe Saturday is a six okay, so these numbers here are going to go straight into , now vector right, the vector that we're going to be feeding into our neural net right 22, 1-3, 21.7. Okay well see in a moment, we'll actually will normalize them, but more or less, but this categorical variable we're not. We need to put it through an embedding right, so we'll have some embedding matrix right of if there are seven days by, I don't know, maybe dimension for embedding okay, and so this will look up the sixth row to get back the four items right, and so This is going to turn into length four vector, which will then add here. Okay, so that's how our continuous and categorical variables they're going to work.

So then, all of our categorical variables will turn them into pandas categorical variables in the same way that we've done before and then we're going to apply the same mappings to the test set right. So if Saturday is a six in the training set, this apply. Cats makes sure that Saturday is also a six and the test set for the continuous variables make sure they're all floats, because pay torch expects everything to be a float. So then, this is another little trick that I use both of these cells to find something called joined, sent one of them defines them as the whole training set. One of them defines them as a random subset all right, and so the idea is that I do all of my work on the sample make sure it all works well play around with different hyper parameters and architectures, and then, unlike okay, I'm very happy with this. I then go back and run this line of code to say: okay now make that make the whole dataset be the sample and then rerun it okay. So this is a good way again similar to that. What I showed you before it lets. You use the same cells in your notebook to run first of all on the sample and then go back later and run it on the full data set. Okay. So now that we've got that join samp, we can then pass it to proc DF, as we've done before, to grab the dependent variable to deal with missing values, and in this case we pass one more thing, which is du scale, equals true de scale equals true.

We'll subtract the mean and divide by the standard deviation, and so the reason for that is that if our first layer, you know it's just a matrix model play right. So here's our set of weights and our input is like I don't know, it's got something with just like 0.001 and then it's got something like which is like 10 to the 6 right and then our weight matrix has been initialized

to be like random numbers between 0 & amp 1 right so got like 0.6 0.1, etc. Then basically, this thing here is going to have gradients that are nine orders of magnitude bigger than this thing here, which is not going to be good for optimization, okay, so by normalizing everything to be mean of 0 standard deviation of 1 to start with, then that Means that all of the gradients are going to be. You know on the same kind of scale. We didn't have to do that in random forests, because in random forests we only cared about the sort order. We didn't care about the values at all right, but in that with linear models and things that are built out of layers of lynnium, like ie neural nets, we care very much about the scale. Okay, so do scale equals true normalizes our data for us now, since it normalizes our data for us, it returns one extra object, which is a mapper, which is an object that contains for each continuous variable.

What was the mean and standard deviation? It was normalized with the reason being that we're going to have to use the same know mean and standard deviation on the test set correct, because we need our test set in our training set to be scaled in the exact same way. Otherwise, they're going to have different meanings: okay and so these details about making sure that your tests and training set have the same categorical, codings, the same missing value replacement and the same scaling. Normalization are really important to get right, because if you don't get it right, then your test set is: you know not gon na work at all. Okay. But if you follow these steps, you know it'll work, fine. We also take the log of the dependent variable and that's because, in this capital, competition, the evaluation metric was root, mean squared percent error, so root mean squared percent. Error means we're being penalized based on the ratio between our answer and the correct answer. We don't have a loss function in pie, chart called root, mean square percent error. We could write 1 but easier is just to take the log of the dependent, because the difference between logs is the same as the ratio. Okay. So, by taking the log, we kind of get that for free, you'll notice, like the vast majority of regression. Competitions on Kaggle use, either root mean squared percent error or a root mean squared error of the log as their evaluation metric. And that's because in real world problems most of the time we care more about ratios than about raw differences right.

So if you're designing your own project, it's quite likely that you'll want to think about using log of your dependent variable. So then we create a validation set and, as we've learned before most of the time, if you've got a problem involving a time component, your validation set probably wants to be the most recent time period rather than a random subset. Okay. So that's what I do here when I finished modeling and I found an architecture and a set of hyper parameters and a number of epochs and all that stuff that works really. Well. If I want to make my model as good as possible, I'll retrain on the whole thing right, including the validation set right now currently at least past AI assumes that you do have a validation set. So my kind of hacky workaround is to set my validation set to just be one index, which is the first row. Okay, and that way, like all the code keeps working, but there's no real validation set. So obviously, if you do this, you need to make sure that your final training is like the exact same hyperparameters, the exact same number of epochs, exactly the same as the thing that worked, because you don't actually have a proper validation set now to check against. I have a question regarding get elapsed, function which we discussed before so in get elapsed, function we are trying to find.

When is the next holiday like? When will the next one ready come? How many, how many days away so every year the holidays are more or less fixed like there will be holiday on 4th of July 21st assemble and there is hardly any change. So can't we just look from previous years and just get a list of all the holidays that are going to occur this year. Maybe I mean in this case I guess like that's, not sure of promo right and some holidays change like Easter.

You know so like this this this way I get to write one piece of code that works for all of them. You know - and it doesn't take very long to run so yeah, so there might be ways if you're, if your data set was so big that this took too long. You could maybe do it on one year and then kind of somehow copy it, but yeah in this case, so those no need to - and I don't you know - I always value my time over my computer's time. So I try to keep things as simple as I can. Okay, so now we can create our model and

6. 00:32:30

- **How to create our Deep Learning algorithm (or model), using 'ColumnarModelData.from_data_frame()'**
- **Use the cardinality of each variable to decide how large to make its embeddings.**
- **Jeremy's Golden Rule on difference between modern ML and old ML:**
- **"In old ML, we controlled complexity by reducing the number of parameters.**
- **In modern ML, we control it by regularization. We are not much concerned about Overfitting because we use increasing Dropout or Weight-Decay to avoid it"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So to create our model, we have to create a model data object, as we always do with fastai. So a columnar model data object is just a data, a model data object that represents a training set, a validation set and an optional test set of standard columnar. You know structured data, okay, and we just have to tell it which of the variables should we treat as categorical, okay and then pass in our data friends. So for each of our categorical variables here is the number of categories it has. Okay, so for each of our embedding matrices, this tells us the number of rows and that embedding matrix, and so then we define what embedding dimensionality we want. If you're doing like natural language processing, then the number of dimensions you need to capture all the nuance of what a word means and how it's used has been found empirically to be about 600. It turns out that when you do NLP models with embedding matrices that are that are smaller than 600, you don't get as good a results as you do. If you do, if there's the size 600 beyond 600, it doesn't seem to improve much. I would say that human language is one of the most complex things that we model, so I wouldn't expect you to come across many if any categorical variables that need embedding matrices with more than 600 dimensions. At the other end, you know some things may have pretty simple kind of causality right.

So, for example, let's have a look state holiday, you know maybe, if something's a holiday, then it's just a case of like ok at stores that are in the city. There's some behavior there's doors that are in the country, there's some other behavior and that's about it. You know like maybe it's a pretty pretty simple relationship so like. Ideally, when you decide what imbedding size to use, you would kind of use your knowledge about the domain to decide like how complex is the relationship, and so how big embedding do I need right in practice. You almost never know that right, like you, only know that, because maybe somebody else has previously done that research and figured it out like in NLP. So in practice you probably need to use some rule of thumb, okay and then having tried to a rule of thumb, you could then maybe try a little bit higher and a little bit lower and see what helps so, it's kind of experimental. So here's my rule of thumb, my rule of thumb, is look at how how many discrete values the category has ie the number of rows in the embedding matrix and make the dimensionality of the embedding half

of that. Alright. So if a day of week, which is the second one, eight rows and four colors so here it is there right the number of categories divided by two. But then I say: don't go more than 50 right, so here you can see for store.

There's a thousand stores only have a dimensionality of 50. Why 50? I don't know it seems to work okay, so far like you may find you need something a little different actually for the ecuadorian groceries competition. You know I haven't really tried playing with this, but I think we may need some larger, embedding sizes, but it's something to feel a bit princess. Can you pass that left social variables, the cardinality size becomes larger and larger you're creating more and more like. I think we collide in very much seas on you, therefore, massively risking overfitting, which is just using so many parameters of the model, can never possibly capture all that variation. Less. Your data is actually huge, that's a great question, and so let me remind you about my kind of like golden rule, with the difference between modern machine learning and old machine learning, an old machine learning. We control complexity by reducing the number of parameters in modern machine learning. We control complexity by regularization. So short answer is not I'm not concerned about overfitting, because the way I avoid overfitting is not by reducing the number of parameters, but by increasing my dropout or increasing my weight. Okay, okay. Now, having said that, like there's no point using more parameters for a particular embedding than I need like because regularization like is penalizing a model by giving it like more random data or by actually penalizing weights, so we like we'd rather not use more than we have To but they're kind of, my general rule of thumb for designing an architecture is, to you know, be generous on the side of the number of parameters but yeah in this case.

If, after doing some work, we kind of felt like you know what the store doesn't. Actually seem to be that important, then I might manually go and make change this to make it smaller, but you know or if I was really finding there's not enough data here, I'm either overfitting or I'm using more regularization uncomfortable with again. You know, then you might go back, but I would always start with like being generous with parameters and yeah. In this case, this model turned out pretty good okay. So now we've got a list of tuples containing the number of rows and columns of each bar. Embedding matrices, and so when we call get learner to create our neural net. That's the first thing we pass in right is how big is each of our embeddings okay, and then we tell it how many continuous variables we have. We tell it how many activations to create for each layer and we tell it what dropout to use page layer, okay and so then we can go ahead and call fit okay. So then we fit for awhile and we're

7. 00:39:20

- **Checking our submission vs Kaggle Public Leaderboard (not great), then Private Leaderboard (great!).**
- **Why Kaggle Public LB (LeaderBoard) is NOT a good replacement to your own Validation set.**
- **What is the relation between Kaggle Public LB and Private LB ?**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Kind of getting something around the the point, one mark all right, so I tried running this on the test set and I submitted it to kaggle during the week actually last week, and here it is okay. Private score. 107. Public score. 103. Okay, so let's have a look and see how that would go.

So when I was seven private, 103 public's. Let's start on public, which is 103 not there out of 3000. I've got to go back a long way here. It is 103. Okay, three hundred and fortieth yeah - that's not good! So on the public leaderboard three hundred and fortieth: let's try the private leader board, which is 107, oh fifth, so, like hopefully, you are now thinking. Oh, there are some chemical competitions finishing soon which I entered, and I spent a lot of time trying to get good results on the public leaderboard. I wonder if that was a good idea and the answer is no. I want write the cavil public leaderboard is not meant to be a replacement for your carefully developed validation set okay. So, for example, if you're doing the iceburg competition right, which ones are ships which ones icebergs, then they've actually put something like 4,000 synthetic images into the public, leaderboard and money into the private leaderboard okay. So this is one of the really good kind of things that tests you out on Kegel is like.

Are you creating a good validation set, and are you trusting it right? Because if you're trusting your leaderboard feedback more than your validation feedback, then you may find yourself in three hundred and fiftieth place when you thought you're in fifth right. So in this case, we actually had a pretty good validation set right because, as you can see, it's saying like somewhere around 0.1 and we actually did get somewhere around 0.1 okay, and so in this case the validation set. That's re that publicly the board in this competition was entirely useless. Yeah. Can you place the box please? So in regards to that, how much does the top of the public leaderboard actually correspond to the top of a privately reward? Because in the in the churn prediction challenge there's like for people who are just completely above everyone else, that's it, it totally depends. You know like if they randomly sample the public and private leaderboard, then it should be extremely indicative right, but it might not be right. So, in this case, okay - let's crushed - oh here - comes so in this case the person who was second on the public leader. What did end up winning s? Dnt came seventh right, so in fact you can see the little green thing here right. Where else this guy jumped 96 places, if we had entered with the neural net we just looked at, we would have jumped 250 places, so it yeah, it just depends, and so often like you can figure out where the the public leaderboard, like sometimes they'll, tell you The public leaderboard was randomly sampled, sometimes they'll tell you it's not generally.

You have to figure it out by looking at the correlation between your validation set results and the public leaderboard results to see how well they're correlated. Sometimes, if, like 2 or 3, people are way ahead of everybody else. They may have found some kind of leakage or something like that, like that's, often a sign that there's some trick. Okay, so that's Russman, and that brings us to the end of all of our material. So let's come back after the break and do a quick review and then we will talk about ethics and machine level. So let's come

8. 00:44:15

- **Course review (lessons 1 to 12)**
- **Two ways to train a model: one by building a tree, one with SGD (Stochastic Gradient Descent)**
- **Reminder: Tree-building can be combined with Bagging (Random Forests) or Boosting (GBM)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Back in five minutes, so we've learnt two ways to train a model. One is by building a tree and one is with SGD, okay, and so the SGD approach is a way we can train a model which is a linear model or a stack of linear layers. With nonlinearities between them, whereas tree building specifically, will give us a tree okay and then tree building, we can combine with bagging to create a random forest or with boosting to create a GBM or various other slight variations, such as extremely randomize trees. So it's worth like reminding ourselves of like what these things do. So, let's, let's look at some data, so if we've got some data like so actually, let's look specifically, let's look specifically a categorical data right, okay, so categorical data. There's a couple of possibilities of what categorical data might look like it could be like. Okay. So let's say we got zip code like so: we've got line for double O three: here's our zip code right and then we've got like sales right and it's like 50 and like nine four one, three one sales or twenty, two and so forth. All right! So we've got some categorical variable, so there's a couple:

9. 00:46:15

- **How to represent Categorical variables with Decision Trees**
- **One-hot encoding a vector and its relation with embedding**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Of ways we could represent that categorical variable one would be just to use the number right and like maybe it wasn't a number to start. You know, maybe it wasn't a number at all. Maybe a categorical variable is like San Francisco New York, Mumbai and Sydney right, but we can turn it into a number just by like arbitrarily deciding to give them numbers right so like it ends up in your number. So we could just use that kind of arbitrary number. So if, if it turns out that zip codes that are numerically next to each other have somewhat similar behavior, then the zip code versus sales chart might look something like this, for example right or alternatively, if the zip code versus sales. Like sorry, if the two zip codes next to each other didn't have any ways similar similar sales behavior, you would expect to see something that looked more like this, like kind of just all over the place right, okay, so there are kind of two possibilities. So what a random forest would do if we had just encoded zip in this way, is it's gon na say alright. I need to find my single best split point. Okay, the split point: that's going to make the two sides have as smaller standard deviation as possible or mathematically equivalently have the lowest root mean squared error. So, in this case, it might pick here as our first bit point because on this side, there's one average and on the other side there's the other average okay and then, for its second split point, it's going to say: okay, how do I split this and it's Probably going to say, I would split here right because now we've got this average versus this average right and then finally, it's going to say: okay, how do we split here and it's going to say, okay I'll spit there right so now, I've got that average and Average, okay, so you can see that it's able to kind of hone in on the set of spits it needs, even though it kind of doesn't greatly top down one at a time right. The only reason it wouldn't be able to do this as if like it was just such bad luck, that the two halves were kind of always exactly balanced right.

But even if that happens, it's not going to be the end of the world it'll spit on something else, some other variable and next time around. You know it's very unlikely that it's still going to be exactly balanced in both parts of the tree, all right. So in practice this works just fine. In the second case right, it can do exactly the same thing right, it'll say like okay, which is my

best first split right, even as though there's no relationship between one zip code and its neighboring zip code. Numerically, we can still see here if it if it's bits here right, there's the average on one side and the average on the other side is probably about here. Okay and then, where would it split next, probably here all right because here's the average on one side? Here's the average on the other side, all right so again can do the same thing right. It's going to need more splits because it's going to end up having to kind of narrow down on each individual large that code in each individual, small, zip code. But it's still going to be fine, okay, so when we're dealing with building decision, trees for random forests or gbm's or whatever, we tend to encode our variables, just as ordinals okay, on the other hand, if we are doing a a neural network or like a simplest Version like a linear regression or logistic regression, the best it could do is that right, which is no good at all and did over this one.

It's going to mean like that. Okay, so an ordinal is not going to be a useful encoding for a linear model or something that stacks linear and nonlinear models together. So instead what we do is we create a one, hot encoding right so we'll say, like you know, he is zero. One! Zero! Zero zero here: zero 100 - he is Oh 100, one okay, and so with that encoding that can effectively create like a little histogram right, where it's going to have a different coefficient for each level. Okay, and so that way it can do exactly what it needs to do. Can you pass that back? Please, at what point does that become like too tedious for your system, or does it not pretty much, never yeah, because remember in real life, we don't actually. Actually, we don't actually have to create that matrix. Instead, we can, just you know, have the four coefficients right and just do an index lookup to grab the second one, which is mathematically equivalent to x on the one pot encoding. Okay. So so that's no problem. One thing to mention: you know: I know you guys have kind of been taught quite a bit of more like analytical solutions to things and in analytical solutions to like a linear regression, you get, you can't solve something with this amount of collinearity, in other words, Sydnee. Something you know something is certainly if it's not Mumbai or New York or San Francisco, so in other words, there's a hundred percent collinearity between the forth of these classes versus the other three.

And so, if you try to solve a linear aggression, analytically, that way the whole thing falls. Apart now note with SGD, we have no such problem. Okay, like SGD, why would it care right we're just taking one step along the derivative here cares a little right because, like in the end, the main problem with collinearity is that there's an infinite number of equally good solution right, so, in other words, we could increase All of these and decrease this or decrease all of these and increase this and they're going to balance out right and when there's an infinitely large number of good solutions. It means there's a lot of kind of flat spots in the Loess surface and it can be harder to optimize all right. So that's a really easy way to get rid of all of those flat spots, which is to add a little bit of regularization. So if we added a little bit of a little bit of weight decay like one enix, seven, even then that basically says these are not all equally good anymore. The one which is the best is the one where the parameters are the smallest and the most similar to each other and so that'll again move it back to being a nice loss function. Yes, could you just clarify that point you made about why? What hard coding wouldn't be that sure if we have a one hot encoded vector right and we are multiplying it by a set of coefficients right, then that's exactly the same thing as simply saying: let's grab the thing where the one is right, so in other words, If we had stored this as a zero, you know - and this one has a one - and this one is a two right.

Then it's exactly the same as just saying: hey look up that thing in the array, okay, and so we call that version and imbedding right. So an embedding is a model. Clica is a weight matrix. You can multiply by a one-pot encoding and it's just a computational shortcut, okay, but it's

mathematically the same so there's a key difference. So the first you know key difference between like solving linear type models. Analytically versus with SGD with SGD, we don't have to worry about the arity and stuff, or at least not nearly to the same degree, and then the difference between solving a linear or single layer or multi-layer model with SGD versus a trinny. A tree is going to be like it's going to complain about less things right, so in particular, you can just use ordinals as your categorical variables and as we learnt just before. We also don't have to worry about normalizing continuous variables for a tree, but we do have to worry about it for these SGD change models. So then we

10. 00:55:50

- **Interpreting Decision Trees, Random Forests in particular, with Feature Importance.**
- **Use the same techniques to interpret Neural Networks, shuffling Features.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Also learnt a lot about interpreting random forests in particular, and if you're interested, you may be interested in trying to use those same techniques to interpret neural nets right. So if you want to know which of my features are important in a neural net, you could try the same thing. Try shuffling each column in turn and see how much it changes your accuracy, okay and that's going to be your feature importance for your neural net and then, if you really want to have fun recognize, then that shuffling that column is just a way of calculating how Sensitive the output is to that input which, in other words, is the derivative of the output with respect to that input, and so, therefore, maybe you could just ask pipe torch to give you the derivatives with respect to the input directly and see. If that gives you the same kind of answers, okay, you could do the same kind of thing for partial dependence plot you could try. You know doing the exact same thing with your neural net, replace everything in the column with the same value. Do it for 1960. 1961, 1962 plot that that I don't know if anybody who's done these things before, not because it's rocket science, but just because I don't know, maybe no one thought of it or it's not in our library. I don't know, but if somebody tried it, I think you should find it useful. It would make a great blog post, maybe even the paper if you wanted to take it a bit further.

So there's a thought that something you can do so those most of those interpretation techniques are not particularly specific to random forests. Things like the tree interpreter certainly are because they're all about like what's inside the tree, can you pass? It occur we're applying for interpreter for neuron. That's how are we going to make inference out of activations that the path follows for example? So how are we going to like in three interpreter we are like when looking at we are looking at the parts and their contributions of the features. In this case, it will be same with activations. I guess the contributions of each activation on their path, yeah, baby. No, I haven't thought about it. How can we like make in front of the activations? So I'd be careful to say the word inference, because no people normally is the word inference specifically domain. The same is like a test a test time. Okay prediction you make like make some kind of interrogate the model. Yes yeah, I'm not sure we should think about that. Actually, Hinton and one of his students just published a paper on how to approximate a neural net with a tree for this exact reason, which I haven't read the paper. Yet could you pass that so in linear regression and traditional statistics, like one of the things that we focused on, was statistical significance of like the changes and things like that, and so

when thinking about a tree interpreter or even like the waterfall chart, which I guess Is just a visualization um? I guess where does that fit in like

11. 00:59:00

- **Why Jeremy usually doesn't care about 'Statistical Significant' in ML, due to Data volume, but more about 'Practical Significance'.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Because we can see like oh now, this looks important in the sense that it causes large changes. But how do we know that it's like traditionally statistically significant or anything about yeah? So, most of the time, I don't care about the traditional statistical significance and the reason why is that, nowadays, the main driver of statistical significance is data volume, not kind of practical importance and nowadays most of the models you build will have so much data that, like Every tiny thing will be statistically significant, but most of them won't be practically significant, so my main focus therefore, is practical significance, which is: does the size of this influence impact? Your business? You know statistical significance. Only you know like it was much more important when we had a lot less data to work with. If you do need to know statistical significance, because, for example, you have a very small data set because it's like really expensive to label or hard to collect or whatever or it's a medical data set for a rare disease. You can always get statistical significance by bootstrapping, which is to say that you can randomly resample. Your data set a number of times train your model a number of times, and you can then see the actual variation in predictions. Ok, so that's that's with bootstrapping. You can turn any model into something that gives you confidence intervals.

There's a paper by Michael Jordan, which has a technique called the bag of little bootstraps, which actually kind of takes, takes this a little bit further and well worth reading if you're interested, actually positive prints. So you say we don't need one hold encoding matrix in if you're doing random forests or if you are doing any cleavage models. What will happen if we do that and how bad can a model be? If you trigger a one horn, coding yeah, we actually did do it. Remember we had that like maximum category size and we did create one-hot encodings and the reason why we did it was that then our feature importance would tell us the importance of the individual levels and our partial dependence plot. We could include the individual levels, so it doesn't necessarily make the model worse. It may make it better, but it probably won't change it much at all. In this case, it hardly changed it. This is something that we have noticed on real data, also that if cardinality is higher, let's say 50 levels and if you do one hot encoding, the random forest performs very badly yeah yeah, that's right! If the cab now that's why we have that in that's. Why, in fast I we have that like maximum categorical size, you know because at some point you're one hot encoded variables become too sparse right.

So I generally like cut it off at six or seven, also because, like when you get past, that it kind of becomes less useful, because the feature importance, there's going to be too many levels to really look at. So can it not just not look at those levels which are not important and just gives those significant features as important yeah yeah I mean it's, it's it's it'll be okay. You know it's just like once. The cardinality increases to high you're just you're, just splitting your data up, you know too much basically, and so in practice, your ordinal version is likely to be it's likely to be better okay, so yeah, so little, there's no time to you're kind of review everything. But I

think that's the kind of key concepts and then, of course, remembering that you know the embedding matrix that we can use is likely to have more than just one coefficient will actually have a dimensionality of a few coefficients which isn't going to be useful. For most linear models, but once you've got multi-layer models, that's now creating a representation of your category, which is kind of quite a lot richer, and you can

12. 01:03:10

- **Jeremy talks about "The most important part in this course: Ethics and Data Science, it matters."**
- **How does Machine Learning influence people's behavior, and the responsibility that comes with it ?**
- **As a ML practicioner, you should care about the ethics and think about them BEFORE you are involved in one situation.**
- **BTW, you can end up in jail/prison as a techie doing "his job".**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Do a lot more with it. Let's now talk about the most important bit, we started off early in this course talking about how actually a lot of machine learning is kind of misplaced people. Focus on predictive accuracy, like Amazon, has a collaborative filtering algorithm for recommending books and they end up recommending the book which it thinks you're most likely to write highly, and so what they end up doing is probably recommending a book that you already have or that you Already know about and would have bought anyway right, which isn't very valuable. What they should instead have done is to figure out like which book can I recommend that would cause you to change your behavior right, and so that way we actually maximize our lift in sales. Due to recommendations - and so this idea of, like the difference between optimizing influencing your actions, were suggest of improving predictive accuracy, improving predictive accuracy is a really important distinction which, like very rarely discussed in academia or industry, kind of crazy enough. It's more discussed in industry. It's particularly ignored in most of academia right, so it's a really important idea, which is that in the end that our idea, the goal of your model, presumably is to influence behavior, okay and so and remember.

I actually mentioned a whole paper right have about this, where I introduced this thing called the drivetrain approach, where I talk about like ways to think about how to incorporate machine learning into like how do we actually influence behavior? So you know that's a starting point, but then the next question is like okay: if we're trying to influence behavior, what kind of behavior should we be influencing and how and what might it mean when we start influencing behavior okay, because, like nowadays like a lot of The companies that you're going to end up working at are bigger as companies and you'll be building stuff that can influence millions of people right. So what does that mean? So I'm? Actually, I'm not going to tell you what it means because, like I don't know, all I'm going to try and do is make you aware of some of the issues right and and make you believe, two things about them. First, that you should care right and second, that they're big current issues right. The main reason I want you to care is because I want you to want to be a good person and show you that, like not thinking about these things will make you a bad person. But if you don't find that convincing, I will tell you this. Volkswagen were found to be cheating on their emissions tests. The person who was sent to jail for it was the programmer that implemented that piece

of code.

They did exactly what they were told to do right, and so, if you're coming in here thinking, hey I'm just a techie. You know I'll just do what I'm told right. That's that's my job is to do what I'm told I'm. If you do that, you can be sent to jail for doing what you're told okay, so so a don't just do what you're told, because you can be a bad person and B you can go to jail. Okay, second thing to realize is, in the heat of the moment, you're in a meeting with twenty people at work and you're all talking about how you're going to implement you know this new feature and everybody's discussing it and there's some cut. You know and everybody's like we could do this and here's a way of modeling it and then we can implement it and here's these constraints and there's some part of you. That's thinking, I'm not sure we should be doing this right. That's not the right time to be thinking about that, because it's really hard so like step up there and say. Excuse me, I'm not sure this is a good idea. You actually need to think about how you would handle that situation ahead of time. Okay, so I want you to like think about about these issues now right and realize that by the time you're in the middle of it right, you might not even realize it's happening. You know like they'll, just it'll just be a meeting like every other meeting, and a bunch of people will be talking about how to solve this technical question.

Okay and you need to be able to recognize like. Oh, this is actually something with ethical implications, so Rachel actually wrote all of these slides. I'm sorry. She can't be here to present this because, like she's studied this in depth - and you know she's actually being in in in difficult environments herself, where she's kind of seen these things happening, you know, and we know how hard it is right. But let me

13. 01:08:15

- **IBM and the "Death's Calculator" used in gas chamber by the Nazis.**
- **Facebook data science algorithm and the ethnic cleansing in Myanmar's Rohingya crisis: the Myth of Neutral Platforms.**
- **Facebook lets advertisers exclude users by race enabled advertisers to reach "Jew Haters".**
- **Your algorithm/model could be exploited by trolls, harassers, authoritarian governements for surveillance, for propaganda or disinformation.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Give you a sense of like what happens right so so engineers trying to solve engineering problems. Is you know and causing problems is not a new thing right. So in Nazi Germany IBM that the group known as Hollerith right Hollerith was the original name of IBM and it comes from the guy who actually invented the use of punch cards for tracking the US Census. The first mass wide-scale use of punch cards for data collection in the world right and that turned into IBM, and so at this point, if this this unit at least was still called Hollerith, so Hollerith sold a punch card system to Nazi Germany and so each punch Card would like code. You know this is a Jew, eight, gypsy, twelve general execution for death by gas chamber six, and so here's one of these cards describing the right way to kill these various people right and so a Swiss judge ruled that IBM's technical assistance facilitated the tasks of the Nazis and Commission of their crimes against humanity. This led through the death of something like twenty million civilians. So, according to the Jewish virtual library, where I got these pictures

and quotes from their view is that the destruction of the Jewish people became even less important because of the invigorating nature of IBM's technical achievement only fightin by the fantastical profits to be made right.

So this was a long time ago and you know hopefully you won't end up working at companies that facilitate genocide Ryan, but perhaps you will right because perhaps you'll go to Facebook who are facilitating genocide. Right now, and - and I know people at Facebook who are doing this - and they had no idea - they were doing this right so right now in facebook, the rahega in the middle of a genocide, a Muslim population of Myanmar babies, are being grabbed out of their mother's Arms and thrown into fires people are being killed. Hundreds of thousands of refugees when interviewed the Myanmar generals. Doing this say we are so grateful to Facebook for letting us know about the ringing of fake news that the words they use their finger, fake news that these people are actually not human, that they're actually animals right now, Facebook did not set out to enable the Genocide of their hinga people in Myanmar. No, instead, what happened is they wanted to maximize impression in place right, and so it turns out that for the data scientists at Facebook's, their algorithms kind of learned that if you take the kinds of stuff people are interested in and think them slightly more extreme versions Of that you're actually going to get a lot more impressions and the project managers are saying maximize these impressions and people are clicking and like it, creates this.

This thing right, and so the the the potential implications are extraordinary and global right, and this is something that like is literally happening. You know this is October 2017. Is it's happening now? Okay? Could you pass that back there, so I just want to clarify what was happening here. So it was the facilitation it's like fake news or like inaccurate media yeah. So what happened was. Let me go into it in more detail. So what happened was in mid-2016 Facebook fired its human editors right, so it was humans that decided how to order things on your homepage. Those people got fired and replaced with machine learning, algorithms, and so the machine learning algorithms written by data scientists. Like you, you know they had nice, clear metrics and they were trying to maximize their predictive accuracy and be like okay. We think, if we put this thing higher up than this thing will get more place. Okay, and so it turned out that these algorithms, for putting things on the facebook newsfeed had a tendency to say, like Oh human nature, is that we tend to click on things which, like stimulate our views and are therefore like more extreme versions of things. We already see okay, so so this is great for the kind of Facebook revenue model of maximizing engagement. It looked good on all of their KPIs, and so at the time you know there was some negative press about like you know, I'm not sure that the staff that Facebook's now putting on their trending section is actually that accurate.

That, from the point of view of the metrics that people are optimizing at Facebook, it looked terrific and so way. Back to October 2016 people started noticing some serious problems. For example, it is illegal to target housing to people of certain races in America that is illegal, and yet a news organization discovered that Facebook was doing exactly that right in October 2016. Ok, not because somebody in that data science team said like, let's make sure black people can't live in nice, neighborhoods right, but instead you know they found that their automatic clustering and segmentation algorithm found. There was a cluster of people who didn't like african-americans and that if you targeted them with these kinds of ads, then they would be more likely to select this kind of housing or whatever right. But the interesting thing is that, even after being told about this three times, Facebook still hasn't fixed it right and that is to say these are not just technical issues. They're also economic issues right when you start

saying like the thing that you get paid for, that is ads, you have to change the way that you structure those so that you know you either use more people that cost money or you like a less aggressive on Your algorithms to target people, you know based on like minority group status or whatever you know, that can impact revenues, and so the reason I mention this is you will at likely at some point in your career, find yourself in a conversation where you're thinking, I'm not Confident that this is like morally ok, the person you're talking to is thinking in their head.

This is going to make us a lot of money that, and you just you, don't quite ever manage to have a successful conversation because you're talking about difficult, different things, you know, and so, when you're talking to somebody who may be more experienced and more senior than You and they may sound like they know what they're talking about right, just realized, that their incentives are not necessarily going to be focused on like how do I be a good person. You like they're, not thinking how it might be a bad person, but you know the more time you spend an industry. In my experience, the more desensitized you kind of get to this stuff of, like okay, maybe getting promotions and making money, isn't the most important thing right. So, for example, I've got a lot of friends who are very good at computer vision, and some of them have gone on to create startups that seem like they're, almost handmade to help authoritarian governments surveil their. You know their citizens and when I ask my friends like have you thought about how this could be used in that way? You know they're generally kind of offended that I asked you know, but but I'm asking you to think about this, like you know, wherever you end up working, if you end up creating a start-up like tools can be used for good or for evil right, and so I'm not saying like don't create excellent object, tracking and detection tools from computer vision, because yeah you could go on and use that to create like a much better surgical intervention, robot tool, kit right.

I've just seen like be aware of it, think

14. 01:16:45

- **Runaway feedback loops: when Recommendation Systems go bad.**
- **Social Network algorithms are distorting reality by boosting conspiracy theories.**
- **Runaway feedback loops in Predictive Policing: an algorithm biased by race and impacting Justice.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

About it, talk about it, you know so here's what I'd find like fascinating and there's this really cool thing. Actually, that made up calm did this is from a made-up contour kits online. They they think about this. They actually thought about this. They actually thought you know what, if we built a collaborative filtering system like we learned about in class, to help people decide what meetup to go to. It might notice that on the whole in San Francisco, a few more men and women tend to go to techie meetups, and so it might then start to decide to recommend techie meetups to more men than women, as a result of which more men will go to Techie meet us, as a result of which, when women go to techie meet ups they'll be like. Oh, this is all men. I don't really want to go to tech. He made ups as a result of which the algorithm will get new data, saying that men like taking meetups that right, and so it continues Matt and so like a little a little bit of kind of that initial push from the algorithm can create this runaway feedback. Loop and you end up with, like almost all my old tech meetups, for instance right and so this

kind of feedback loop, is a kind of subtle issue that you really want to think about when you're thinking about like what is the behavior that I'm changing with This algorithm that I'm building so another example, which is kind of terrifying, is in this paper, where the authors describe how a lot of the partment s -- in the US are now using predictive policing, algorithms right.

So where can we go to find somebody who's about to commit a crime, and so you know that the algorithm simply feeds back to you. Basically, the data that you've given it right. So if your Police Department has engaged in racial profiling at all in the past, then it might suggest slightly more often. Maybe you should go to the black neighborhoods to check for people committing crimes right, as a result of which more of your police officers go to the black neighborhoods, as a result of which they arrest more black people, as a result of which the data says that The black neighborhoods are less safe, as a result of which the algorithm says to policeman. Maybe you should go to the black neighborhoods more often and so forth. Right - and this is not like you know, vague possibilities of something that might happen in the future. This is like documented work from top academics who have carefully studied the data and the theory right. This is like serious scholarly work is like no. This is this is happening right now, and so you know again, like I'm sure, all the people that started creating this predictive, policing, algorithm didn't think like how do we arrest more black people right? You know. Hopefully they were actually thinking. Gosh I'd like my children, to be safer on the street. It's how do I create you know a safer society right, but they didn't think about this.

This nasty, runaway feedback loop. So actually this this one about social network algorithms is actually a article in The New York Times recently that one of my friends Renee direst her and she did something that was kind of amazing. She set up a second Facebook account all right, like a fake facebook account, and she was very interested in the anti vex movement at the time. So she started following a couple of anti-vaxxers and visited a couple of anti-vaxxer links and so suddenly her newsfeed starts getting full of anti-vaxxer news along with other stuff, like chemtrails and deep state conspiracy theories and all this stuff, and so she's like starts clicking on those Right and the more she clicked the more hardcore far-out conspiracy, stuff facebook recommended. So now, when Renee goes to that Facebook account, the whole thing is just full of angry crazy, far-out conspiracy, stuff, like that's all she sees, and so if that was your world right, then as far as you're concerned is just like this continuous reminder and proof of Of all this stuff right - and so again, it's like this. This is to answer your question. This is the kind of ran away feedback loop that ends up telling me and my generals, you know throughout their Facebook homepage that reinga

15. 01:21:45

- **Bias in Image Software (Computer Vision), an example with Faceapp or Google Photos. The first International Beauty Contest judged by A.I.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Animals and fake news, and whatever else all right, so you know it's it's a lot of this comes from also from bias right and so like. Let's talk about bias specifically so bias in image, software comes from bias in data and so most of the folks. I know at Google brain building computer vision, algorithms very few of them are people of color, and so when they're

training, the algorithms with you know, photos of their families and friends. They are training them with very few people of color, and so, when face up then decided we're gon na try. Looking at lots of Instagram photos to see which ones are like you know, I've voted the most without them, necessarily realizing it. The answer was, like you know, light colored faces. So then they built a generative model to make you more hot, and so this is the actual photo, and here is the hotter version right. So the harder version is like more white, less nostrils. You know more european-looking, right and so like this did not go down well, to say the least. So like that, so again you know, I don't think anybody at face app said like let's create something that makes people look more white right. They just trained it on a bunch of images of the people that they had around them. Okay - and this has kind of you - know serious commercial implications as well. They had to pull this feature right and they had a huge amount of negative pushback like ads.

A short right - here's another example: Google photos created this photo classifier, airplane, skyscrapers cars, graduation and no guerrillas right. So, like think about how this looks to like most people like most most people, they look at this. They don't know about machine learning, they say what the somebody at Google wrote some code to take black people and call them gorillas. Like that's what it looks like right now. We know: that's not what happened right. We know what happened. Is you know they're a team. You know of folks at Google computer vision experts who have none if a few people of color working in the team built a classifier using all the photos they had available to them, and so when the system came along, came across, you know a person with dark Skin it was like - oh I've, only mainly seen that before, amongst gorillas, so I'll put it in that category right so again, it's the bias and the data creates a bias in the software and again, the commercial implications were very significant. Like Google really got a lot of bad PR as they should this, this was a photo that some you know somebody put in their Twitter feed. They said like look. What look what Google photos just decided to do? You can imagine what happened with the first international beauty contest judged by artificial intelligence, and basically it turns out all the beautiful people of what okay right so like you could kind of see this bias in image software thanks to bias in the data thanks to by Lack of diversity and the

16. 01:25:15

- **Bias in Natural Language Processing (NLP)**
- **Another example with an A.I. built to help US Judicial system.**
- **Taser invests in A.I. and body-cameras to “anticipate criminal activity”.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Team's building it, you see the same thing in natural language. Processing. Alright, so here is Turkish. Oh is the the pronoun in Turkish, which has no gender right. There is no he or versus she right cut him, no okay! No! He visit she but of course, in English we don't really have a widely used and gendered singular pronoun. So Google Translate converts it to this okay. Now there are plenty of people who saw this online and said, like literally so what you know it is correctly feeding back the usual usage in English. Like this, is you know it's it? I know how this is trained. This is like word to Vic vectors. I was trained on Google News corpus, Google books corpus, it's just telling us how things are and like from a point of view, that's entirely true right, like the biased data to create this biased algorithm is the actual data of how people are written books and newspaper Articles for decades, but does that mean that

this is the product that you want to create? You know, does this mean this is the product you have to create, but just because the particular way you've trained the model means it ends up doing this, you know, is this actually the design you want and can you think of potential negative implications and feedback loops? This could create no, and you know if any of these things bother you there now you're lucky you.

You have a new cool engineering problem to work on like how do I create unbiased and RP solutions, and now there are some startups starting to do that and starting to make some money right so like opera, these are opportunities for you. It's like hey. Here's some stuff where people are creating screwed up societal outcomes because of their shitty models like okay. Well, you can go and build something better right. So like another example of the bias in word, l�vesque word: vectors is restaurant reviews ranked Mexican restaurants worse because Mexico, the Mexican words, tend to be associated with criminal words in the u.s. press and books. More often again, this is like a real problem that is happening right now, so you know, Rachel actually did some interesting analysis of just the plain word for backward vectors where she basically pulled them out - and you know looked at these analogies based on some research that Had been done elsewhere, and so you can see like where to Vic like the the vector directions show that father is the doctor is the mother is too nervous. Man is too computer programmer, as women is the homemaker and so forth right so like it's, it's really easy to see. What's in these word vectors - and you know they're kind of fundamental to much of the NLP you're, probably just about all the NLP software we use today, so like here's a great example, so a pro public has actually done a lot of good work in this area.

Judges many judges now have access to sentencing, guideline software and so Sentencing Guidelines. Software says to the judge for this individual. We would recommend this kind of sentence right and now, of course, a judge doesn't understand machine learning so like they have two choices, which is either. Do what it says or ignore it entirely right and some people fall into each category right and so for the ones that fall into the like. Do what it says, category here's what happens for those that were labeled higher risk right, the subset of those that label high risk, but actually turned out not to rear fender quarter of whites and about 1/2 of african-americans. Okay. So, like nearly twice as often right, people who didn't really marked as higher risk, if they're african-american and vice versa, amongst those labeled lower risk, but actually did reoffended to be about half of the whites and only 28 % of the african-americans like so like. This is data which I I would like to think nobody is setting out to create something that does this right. But when you start with bias data right - and you know, the data says that whites and blacks smoke marijuana at about the same rate that blacks are jailed at. I think it's something like five times more often than whites like you know, the nature of the justice system in America, or at least at the moment, is that it's not it's not equal.

It's not fair and therefore the data that's fed into the machine learning model is going to basically support that status quo and then because of the negative feedback loop, it's just going to get worse and worse right now, I'll tell you something else interesting about this one Which research court erred Gong has pointed out is here are some of the questions that are being asked right. So, let's, let's take one, was your father ever arrested right. So your answer to that question: it's going to decide whether you're locked up and for how long now, as a machine learning researcher, do you think that might improve the active accuracy of your algorithm and get you a better r-squared. It could well, but I don't know you know, maybe it does you try it out. So oh I've got a bit of R squared, so does that mean you should use it like? Well, there's another question: like: do you think it's reasonable to lock somebody up for longer because of who their dad was that and yet these are actually the examples of questions

that we are asking right now to offenders and then putting into a machine learning system to Decide what happens to them? Okay, so again like whoever designed this. Presumably they were like laser focused on technical excellence, getting the maximum area under the ROC curve, and I found these great predictors that give me another 0.02 right and I guess didn't stop to think like.

Well, is that a reasonable way to decide who goes to jail for longer so like putting this together? You can kind of see how this can that you know more and more scary, we take a company like taser, right and tasers. Are these devices that kind of give you a big electric shock, basically and tasers, manage to do a great job of creating strong relationships with some academic researchers who seem to say whatever they tell them to say to the extent where now, if you look at the Data it turns out that there's a much higher problem. You know, there's a pretty high probability that if you get tased that you all die, it happens, you know not. Unusually and yet you know the researchers who they've paid to look into this have consistently come back and said: oh no, it was nothing to do with the Taser. The fact that they died immediately afterwards was totally unrelated. It was just a random, you know things things happen, so this company now owns 80 % of the market for body cameras and they've started buying computer vision, AI companies and they're going to try and now use these police body camera videos to anticipate criminal activity. Okay and so like what does that mean right? So is that, like okay, I now have some augmented reality display saying like pays this person because they're about to do something bad, you know. So it's like it's kind of like a whirring direction, and so you know I'm sure nobody who's a data scientist at taser or at the companies that they bought out is thinking like you know, this is the world I want to help create that they could find Themselves in you know, or you could find yourself in the middle of this kind of discussion, where it's not explicitly about that topic, but there's part of you, that's just like wow. I wonder if this is how this could be used right and - and you know I don't know exactly what the right thing to do in that situation is because, like you can ask and of course people gon na be like no.

No, no! No! So it's like you know, are you gon na? You know what what could you do know you could like ask for some kind of written promise. You could decide to leave. You could you know start doing some research into the legality of things to say like? Oh, I would at least protect my own. You know legal situation.

17. 01:34:30

- **Questions you should ask yourself when you work on A.I.**
- **You have options !**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

I don't know like have a think about how you would respond to that. So these are some questions that reaction Rachel created as being things to think about right. So if you're looking at building a data product - or you know using a model like if you're, building machine learning model, is your a reason? Okay, if you're trying to do something right. So what bias may be in that data right? Because, whatever bias is in that data ends up being a bias in your predictions, potentially then biases the actions you're influencing potentially then biases, the data that you come back and you may get a feedback loop. If the team that built it isn't diverse, you know what might you be missing yeah so, for example, one senior executive

at Twitter called the alarm about major Russian bot problems at Twitter way back well before the election. That was the one black person in the exact team, a Twitter, the one and shortly afterwards they lost their job right and so like it definitely having a more diverse team. That means having a more diverse set of opinions and beliefs and ideas and things to look for and so forth. So non diverse teams seem to make more of these bad mistakes. Can we order a code as an open-source check for the different error rates amongst different groups? It's there like a simple rule.

We could use instead, that's like extremely interpretive all and easy to communicate and, like you know, if something goes wrong, do we have a good way to deal with it? Okay, so when, when we've talked to people about this and a lot of people like have come to Rachel and said, like I'm, I'm concerned about something my organization's doing, you know what do I do or I'm just concerned about my toxic workplace? What do I do and very often you know Rachel will say like well, have you considered leaving and they will say all I I don't want to lose my job, but actually, if you can code you're in like 0.3 percent of the population, if you can code And do machine learning you're in probably like 0.01 percent of the population? You are massively massively in demand so, like realistically, you know. Obviously it's an organization does not want you to feel like you're somebody who could just leave and get another job. That's not in your interest in their interest, but that is absolutely true right, and so one of the things I hope you'll leave this course with is, is enough self-confidence to recognize that you have the skills you know to get to get a job, and particularly once You've got your first job. Your second job is an order of magnitude, easier right, and so you know this is important, not just so that you feel like you actually have the ability to act ethically.

But it's also important to realize, like if you find yourself in a toxic environment right which is which is pretty damn common. Unfortunately, like there's a lot of shitty tech cultures, environments, particularly in the Bay Area. Right, you find yourself in one of those environments. The best thing to do is to get the hell out right and, and if you don't have the self-confidence to think you can get another job, you can get trapped right. So you know it's really important. You're really important to know that you are leaving this program with very in demand skills and, particularly after you have that first job you'll nail somebody with in-demand skills and a track record of being employed in that area. Okay, okay, great! So! Yes, this is kind of just a broad question, but what are some things that you know of that people are doing to treat bias in data? You know it's kind of like a bit of a controversial subject at the moment, and there are there are like people are trying to use. Some people try to use an algorithmic approach. You know where they're basically trying to say how can we identify the bias and kind of like subtract it out, but like that, the most effective ways I know of the ones that are trying to treat it at the data level. So, like start with a more diverse team, particularly a team involved, it you know and concludes people from the humanities like sociologists, psychologists, economists, people that understand, feedback, loops and implications for human behavior, and they tend to be equipped with.

You know good tools for kind of identifying and tracking these kinds of problems and so and then kind of trying to incorporate the solutions into the process itself. Let's say there isn't kind of. Like a you know some standard process. I can point you to and say here's how to solve it. You know if there is such a thing we haven't found it yet. You know it requires a diverse team of smart people, aware of the problems and what had of them it's the short answer. How can you pass that back please? This is just kind of a general thing. I guess for the whole class, if you're interested in the stuff that I read a pretty cool book, Jeremy you've probably heard of it weapons of mass destruction by Cathy O'neil. It covers a lot of the same stuff, yeah, just

more topics. Yeah thanks the recommendation, so Kathy's great she's also got a TED talk. I didn't manage to finish the books. It's so damn depressing. I was just like yeah no more but yeah it's it's! It's very good all right. Well, that's it! Thank you, everybody! You know this has been. This has been really intense for me, you know. Obviously this was meant to be something that I was sharing with Rachel, so I've you know, ended up doing one of the hardest things in my life, which is to take two peoples worth, of course, on my own and also look after a sick wife and have A toddler and also do a deep learning course, and also do all this with a new library that I just wrote. So I'm looking forward to getting some sleep, but it's been, it's been totally worth it because you've been amazing, like I'm thrilled with how you've you know, reacted to that kind of.

You know the opportunities I've given you and also to the feedback that I've given you. So, congratulations.