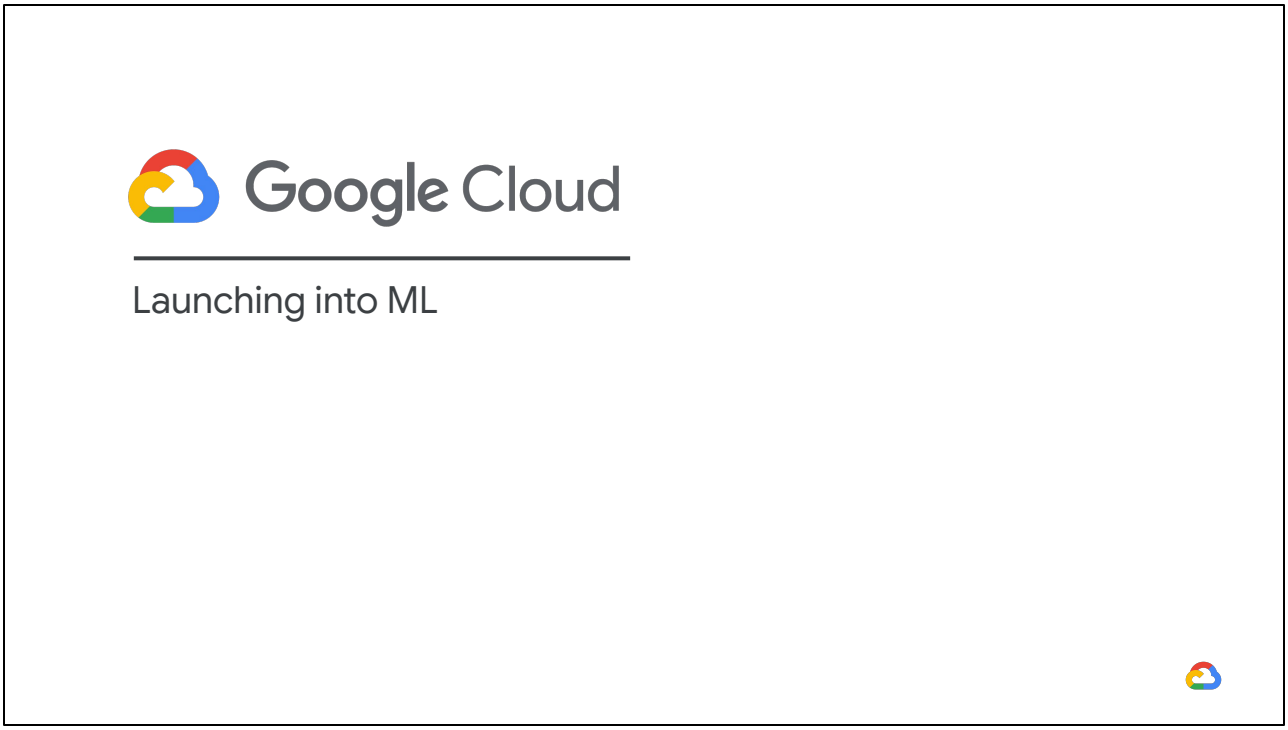


Welcome to Launching into ML.

Agenda

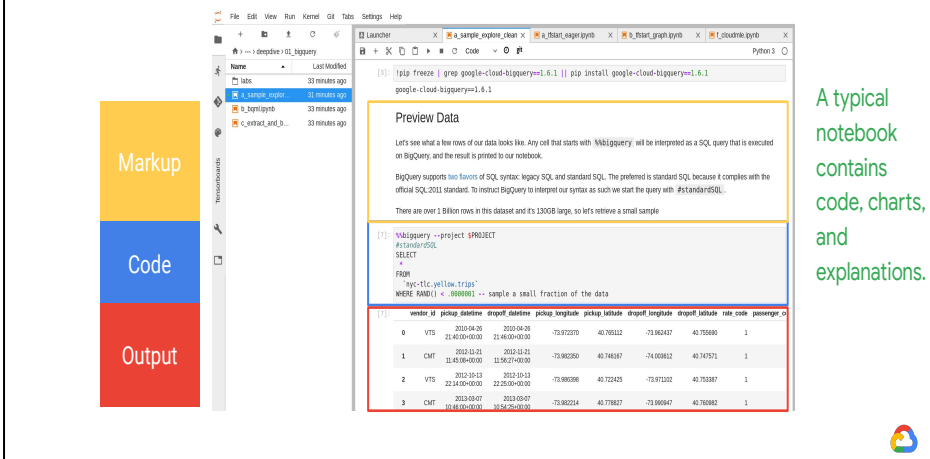
Python notebooks in the Cloud

Supervised Learning

Inclusive ML

Short History of ML

We'll now set up our work environment that we'll be using going forward, deep learning VMs.



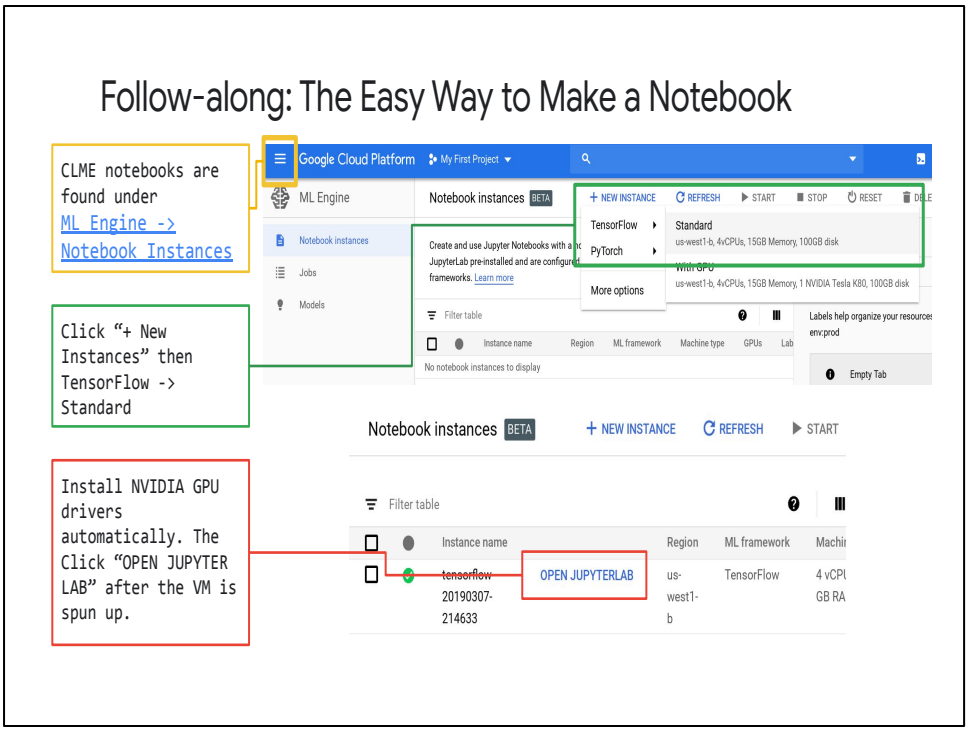
Do you use IPython or Jupyter notebooks today? Increasingly, data scientists work in self-descriptive, shareable, executable notebooks whenever they want to do data analysis or machine learning.

This is what the interface to Jupyter Notebook looks like. Notice how there are code sections interleaved with markup and output. This interleaving is what makes this style of computing so useful.
Data analysis and machine learning are commonly carried out in notebooks like this.

You can execute the code by either clicking the Run button or by pressing Shift + Enter.
Notice that output here isn't just command line output; it's charts as well.
The red section contains markup, so you can explain why you're doing what you're doing.
One of the green sections contains a button for exporting the notebook as a standalone file. The other green section is what you'd click if you wanted to commit your changes to a code repository in GCP. In the lab, we'll show you how to pull from and commit to code repositories.

You can execute the code by either clicking the Run button or by pressing Shift + Enter.
Notice that output here isn't just command line output; it's charts as well.

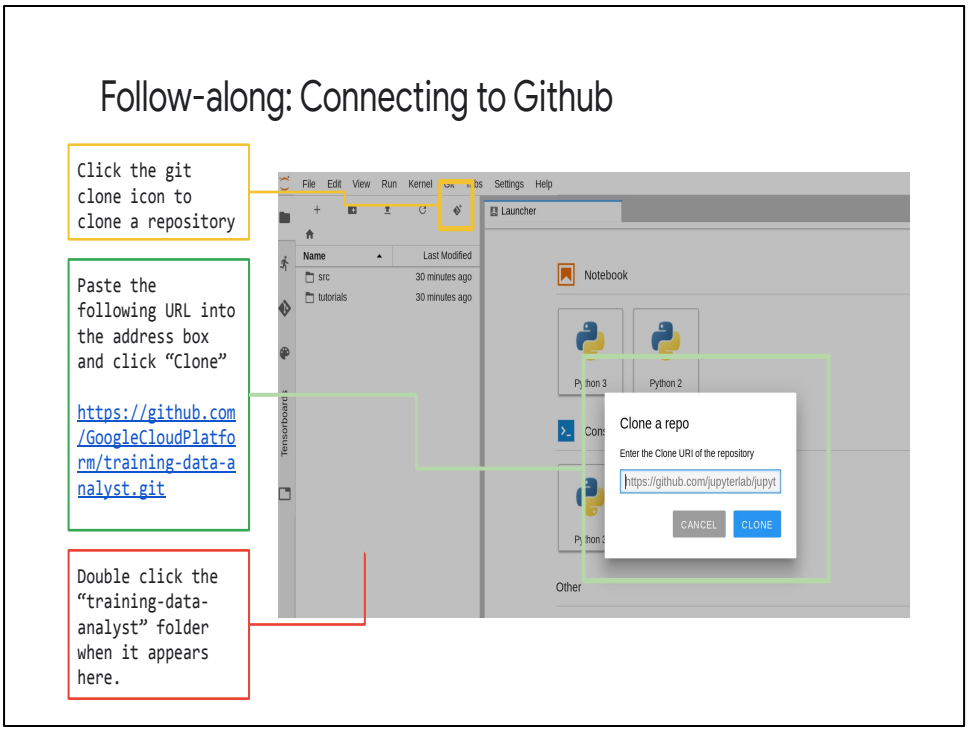
The red section contains markup, so you can explain why you're doing what you're doing.
One of the green sections contains a button for exporting the notebook as a standalone file. The other green section is what you'd click if you wanted to commit your changes to a code repository, in GCP. In the lab, we'll show you how to pull from and commit to code repositories.



After submitting the command line instructions, we can check the progress of our virtual machine by going to Compute Engine -> VM Instances under the menu in the top left corner. There will be a spinning wheel while the VM is booting and a checkmark when it is ready to use. We can click the refresh button every couple of minutes to see if it's ready.

Once it is, we can click the the VM name to look at it's details such as IP addresses, Google Cloud API scopes, and hardware specs. Some of these are editable, but a few of these properties are only editable once the machine is powered off.

One of the VM properties is a proxy-url to access the jupyter notebook. Copy and paste it into a nw chrome tab to see it. As long as you are signed into chrome with your proxy-user-email (which should be your qwiklabs account), you can access this notebook.



Now that we have a notebook, let's fill it with the labs we'll be exploring in this class. On the left hand panel, there's a directory of files in our notebook. Above this directory, the rightmost icon allows us to clone a git repository. Click it and paste the Google Cloud Platform training data analyst url into the box that appears. Then, click "clone".

Afterwards, double click on the new "training-data-analyst" folder when it appears.

We discussed ML as a process and how Google has adopted several philosophical positions that have been crucial to our ML success. What you haven't done yet is dive into what ML is and how it works. That's what you'll do now.

In the next few slides we'll cover:

- Supervised learning, which is one branch of machine learning where you give the model labeled examples of what it should learn.
- A history of ML, to survey the algorithms of the last 50 years, and to understand why neural networks are so prominent at this moment.

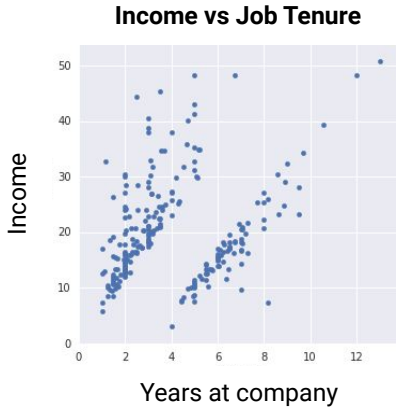
Let's start with Supervised Machine Learning.

Unsupervised and supervised learning are the two types of ML algorithms

Example Model: Clustering

Is this employee on the "fast-track" or not?

In unsupervised learning, data is not labeled.

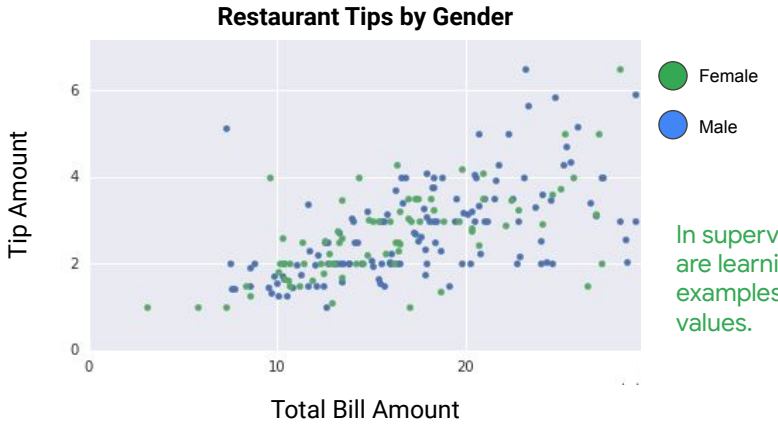


Ryan: Two of the most common classes of machine learning models are supervised and *unsupervised* ML models. The key difference is that with supervised models, we have labels, or in other words, the correct answers to whatever it is that we want to learn to predict.

In unsupervised learning, the data does not have labels.

This graph is an example of the sort of problem that an unsupervised model might try to solve. Here, you want to look at tenure and income, and then group or cluster employees, to see whether someone is on the fast track. Critically, there is no ground truth here; management doesn't, as far as you know, have a big table of people they are going to promote fast, and those they are not going to promote. Consequently, unsupervised problems are all about discovery, about looking at the raw data, and seeing if it naturally falls into groups. At first look, it seems that there are two distinct clusters or groups that I could separate nicely with a line.

Supervised learning implies the data is already labeled



In supervised learning we are learning from past examples to predict future values.

Ryan: In this chapter though, we'll be focused on supervised machine learning problems, like this one. The critical difference is that with supervised learning, we have some notion of a "label," or one characteristic of each data point that we care about a lot.

Typically, this is something you know about in historical data, but you don't know in real time. You know other things, which you call predictors, and you want to use those predictors to predict the thing you don't know.

For example, let's say you are the waiter in a restaurant. You have historical data of the bill amount and how much different people tipped. Now, you are looking at the group sitting at the corner table. You know what their total bill is, but, you don't know what their tip is going to be. In the historical data, the tip is a label. You create a model to predict the tip from the bill amount. Then, you try to predict the tip, in real time, based on the historical data and the values that you know for the specific table.

Regression and classification are supervised ML model types

	total_bill	tip	sex	smoker	day	time
2	16.99	1.01	Female	No	Sun	Dinner
3	10.34	1.66	Male	No	Sun	Dinner
4	21.01	3.5	Male	No	Sun	Dinner
5	23.68	3.31	Male	No	Sun	Dinner
6	24.59	3.61	Female	No	Sun	Dinner
7	25.29	4.71	Male	No	Sun	Dinner
8	8.77	2	Male	No	Sun	Dinner
9	26.88	3.12	Male	No	Sun	Dinner

Option 1
Regression Model
Predict the tip amount

Option 2
Classification Model
Predict the sex of the customer

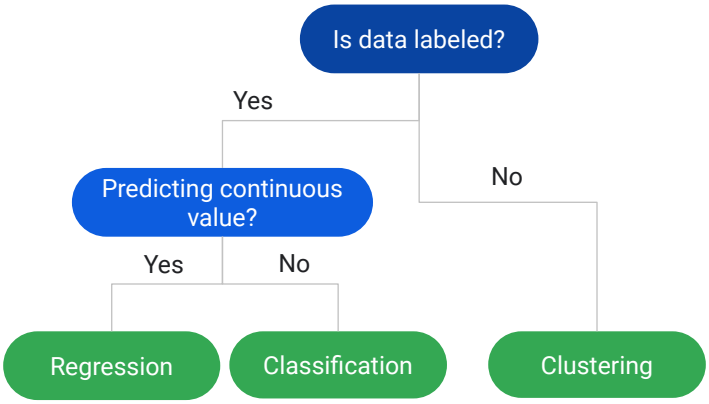
Ryan: Within supervised ML there are two types of problems: regression and classification. To explain them, let's dive a little deeper into this data.

In this dataset of tips, an example dataset that comes with the Python package *seaborn*, each row has many characteristics, such as total bill, tip, and sex. In machine learning, we call each row an "example." You'll choose one of the columns as the characteristic we want to predict, called the "label," and you'll choose a set of the other columns, which are called the "features."

In model option 1, you want to predict the tip amount, therefore the column *tip* is your label. You can use one, all, or any number of the other columns as my features to predict the tip. This will be a regression model because *tip* is a continuous label.

In model option 2, you want to predict the sex of the customer, therefore the column *sex* is the label. Once again, you will use some set of the rest of the columns as your features, to try and predict the customer's sex. This will be a classification model because our label *sex* has a discrete number of values or classes.

The type of ML problem depends on whether or not you have labeled data and what you are interested in predicting



In summary, depending on the problem you are trying to solve, the data you have, explainability, etc. will determine which machine learning methods you use to find a solution.

Your data isn't labeled? You won't be able to use supervised learning then, and will have to resort to clustering algorithms to discover interesting properties of the data.

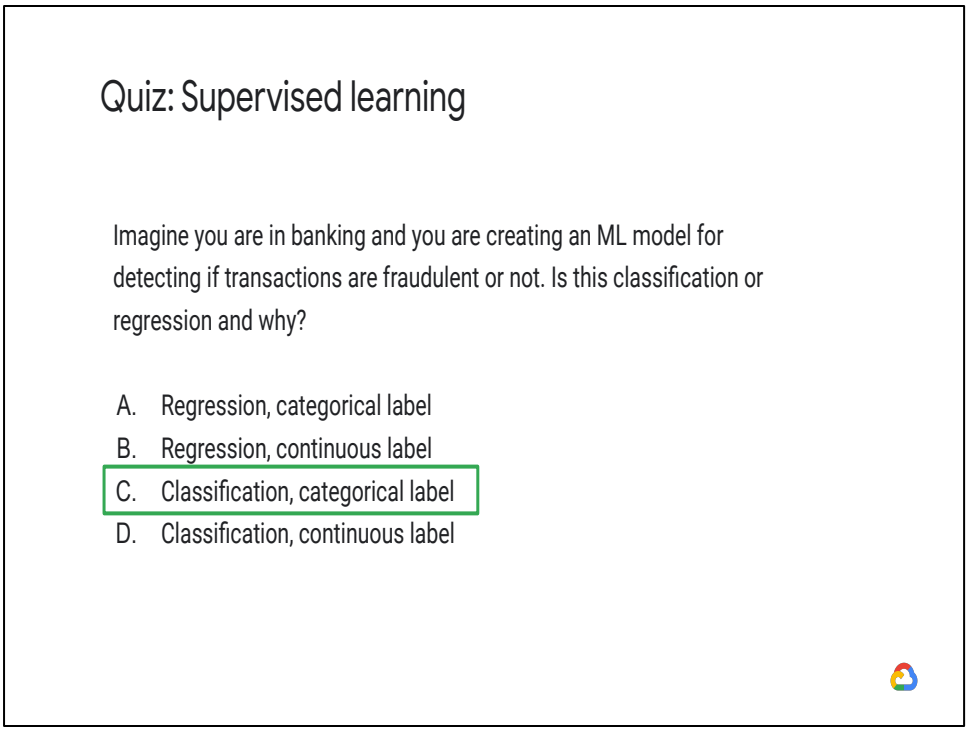
Your data is labeled and the label is dog breed, which is a discrete quantity since there are a finite number of dog breeds? You should use a classification algorithm. If instead the label is dog weight, which is a continuous quantity, you should use a regression algorithm. The label, again, is the thing that you are trying to predict. In supervised learning, you have some data with the correct answers.

Quiz: Supervised learning

Imagine you are in banking and you are creating an ML model for detecting if transactions are fraudulent or not. Is this classification or regression and why?

A. Regression, categorical label
B. Regression, continuous label
C. Classification, categorical label
D. Classification, continuous label

Question: Imagine you are in banking and you are creating an ML model for detecting if transactions are fraudulent or not. Is this classification or regression and why?

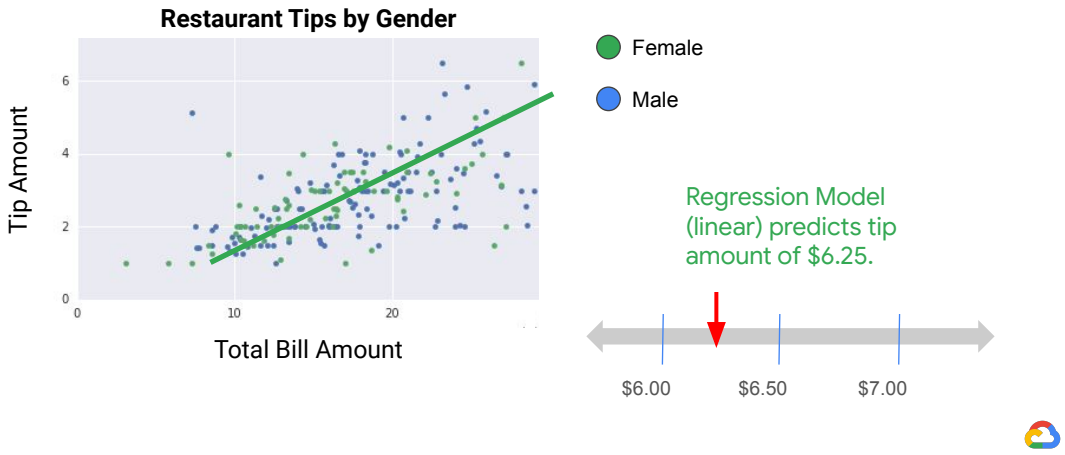


Answer: The correct answer is classification, categorical label. This is a binary classification problem because there are two possible classes for each transaction: fraudulent or not fraudulent. In practice, you may actually have a third class, uncertain. This way depending on your classification threshold it could send any cases that it can't firmly place into the fraudulent or not fraudulent buckets to a human to have a closer look. It is often good practice to have a human in the loop when performing machine learning.

You can eliminate Regression, categorical label and Classification, continuous label because the model types have the opposite label type than they should.

Regression, continuous label at least is a correct pairing however it is incorrect because this is a classification problem so you would not use regression. You could, also create a regression model such as predicting the number of fraudulent transactions, fraudulent transaction amounts, etc.

Use regression for predicting continuous label values



We looked at the tips dataset and said that we could use either the tip amount as the label or the sex of the customer as the label.

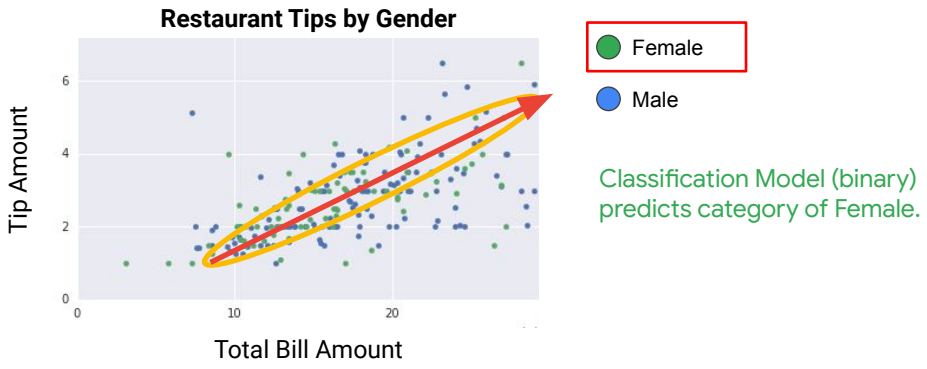
In option 1, we are treating the tip amount as the label and want to predict it, given the other features in the dataset. Let's assume that you are using only one feature -- just the total bill amount -- to predict the tip. Because tip is a continuous number, this is a regression problem. In regression problems, the goal is to use mathematical functions of different combinations of our features to predict the continuous value of our label. This is shown by this line where for a given total bill amount times the slope of the line we get a continuous value for tip amount. Perhaps the average tip rate is 18% of the total bill. Then, the slope of the line would be 0.18.
And by multiplying the bill amount by 0.18, we'll get the predicted tip.

This linear regression with only one feature generalizes to additional features. In that case, we have a multi-dimensional problem, but the concept is the same. The value of each feature for each example is multiplied by the gradient of a hyperplane, which is just the generalization of a line, to get a continuous value for the label.

In regression problems, we want to minimize the error between our predicted continuous value and the label's continuous value, usually using mean squared

error.

Use classification for predicting categorical label values



In Option 2, we are going to treat sex as our label and predict the gender of the customer using data from the tip and total bill. Of course, as you can see from the data, this is a bad idea -- the data for men and women is not really separate -- and we would get a terrible model if we did this. But trying to do this helps me illustrate what happens when the thing you want to predict is categorical and not continuous.

The values that the sex column takes, at least in this dataset, are discrete (male or female). Because sex is categorical and we are using the sex column of the dataset as our label, the problem is a classification problem.

In classification problems, instead of trying to predict a continuous variable, we are trying to create a decision boundary that separates the different classes.
So, in this case, there are two classes of sex; Female and Male. A linear decision boundary would form a line (or a hyperplane in higher dimensions) with each class on either side.
For example, we might say that if the tip amount is > 0.18 * total bill amount, then we predict that the person making the payment was male.
This is shown by the red line.
But that doesn't work very well for this dataset.

Men seem to have higher variability while women tend to tip in a more narrow band. This is an example of a non-linear decision boundary, shown by the yellow ellipse in the graph.

How do we know the red decision boundary is bad, and the yellow decision boundary is better? In classification problems, we want to minimize the error or misclassification between our predicted class and the label's class. This is done usually using cross entropy.

Even if we are predicting the tip amount, perhaps we don't need to know the exact tip amount. Instead, we want to determine whether the tip amount will be high, average, or low. We could define a high tip amount as > 25%, average tip amount as between 15% and 25%, and a low tip amount as being below 15%. In other words, we could discretize the tip amount. And now, predicting the tip amount or more appropriately the tip class becomes a classification problem.

In general, a raw continuous feature can be discretized into a categorical feature.

Later in the specialization, we will talk about the reverse process -- a categorical feature can be "embedded" into a continuous space.
It really depends on the exact problem you are trying to solve and what works best. Machine learning is all about experimentation.

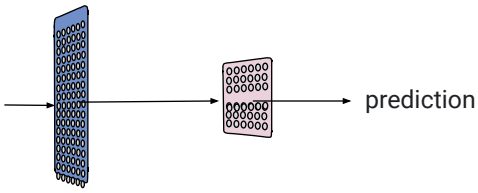
Both of these problem types, regression and classification, can be thought of as "prediction" problems, in contrast to unsupervised problems, which are more like "description" problems.

A data warehouse can be a source of structured data training examples for your ML model

```
SELECT
  gestation_weeks,
  mother_age,
  cigarette_use,
  alcohol_use,
  weight_gain_pounds
FROM
  `bigquery-public-data.samples.natality`
WHERE cigarette_use is not null AND alcohc
```

weight	year	mother_age	gestation_weeks	cigarette_use	alcohol_use
7.86	2003	25	39	false	false
7.5	2003	21	39	false	false
8.06	2004	29	40	false	false
7.56	2004	38	37	false	false
7.06	2003	22	38	false	false

Data on births is sourced from our BigQuery Data Warehouse using SQL.



Now, where does this data come from? The tips dataset is what we call structured data -- consisting of rows and columns -- and a very common source of structured data for machine learning is your data warehouse. Unstructured data is things like pictures, audio, or video.

Here is showing you a natality dataset, a public dataset of medical information. It is a public dataset in BigQuery, and you will use it later in the specialization but for now assume that this dataset is in your data warehouse

Let's say you want to predict the gestation weeks of the baby. In other words, you want to predict when the baby is going to be born.

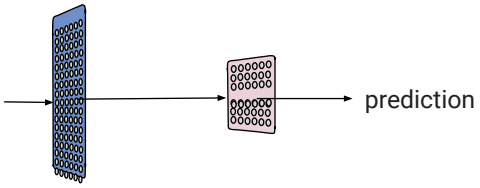
You can do a SQL SELECT statement in BigQuery to create a ML dataset -- you will choose input features to the model, things like mother's age and weight gain in pounds, and the label, gestation weeks.

Because gestation weeks is a continuous number, this is a regression problem.

Making predictions from structured data is very commonplace, and that is what you focus on in the first part of the specialization.

Since baby weight is a continuous value, use regression to predict

weight	year	mother_age	gestation_weeks	cigarette_use	alcohol_use
7.86	2003	25	39	false	false
7.5	2003	21	39	false	false
8.06	2004	29	40	false	false
7.56	2004	38	37	false	false
7.06	2003	22	38	false	false



Weight is stored as a floating point number, representing a continuous (real) value.

Regression DNN Model

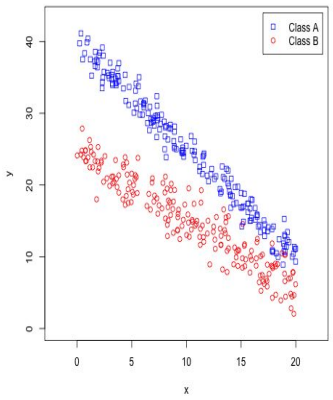
Of course, this medical dataset can be used to predict other things too. Perhaps you want to predict baby weight using the other attributes as our features.
Baby weight can be an indicator of health: when a baby is predicted to have a low birth weight, the hospital will usually have equipment such as an incubator handy. So, it can be important to be able to predict a baby's weight.

The label here would be *baby weight,* and it is a continuous variable. It's stored as a floating point number, which would make this a regression problem.

Quiz: Regression/Classification

Is this dataset a good candidate for linear regression and/or linear classification?

A. Linear classification
B. Both
C. None of the above

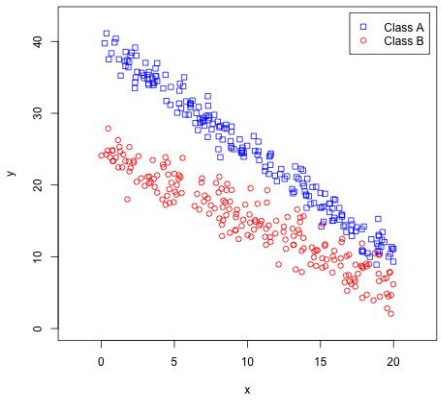


Question: Is this dataset a good candidate for linear regression and/or linear classification?

Quiz: Regression/Classification

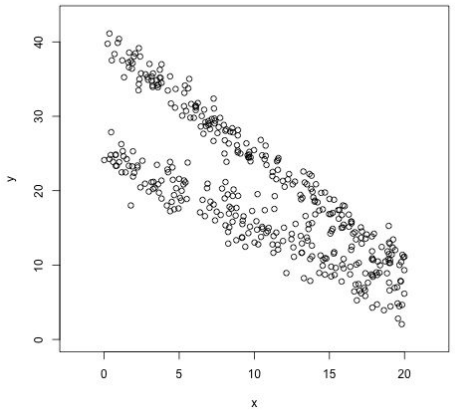
Is this dataset a good candidate for linear regression and/or linear classification?

A. Linear classification
B. Both
C. None of the above



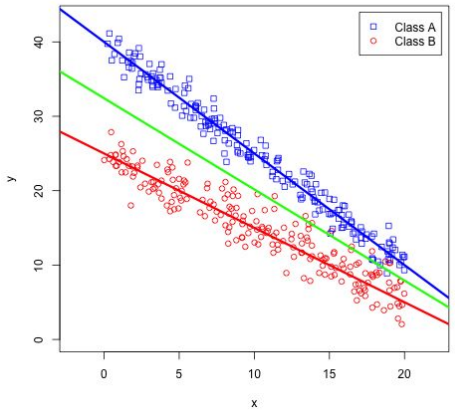
Answer: The correct answer is both. Let's investigate why.

Is this dataset a good candidate for linear regression and/or linear classification?



Let’s step back and look at the dataset with both classes mixed. Without the different colors and shapes to aid us, the data appears to be one noisy line with a negative slope and positive intercept. Since it appears quite linear this will most likely be a good candidate for linear regression, where what you are trying to predict is the value for y.

Is this dataset a good candidate for linear regression and/or linear classification?



Adding the different colors and shapes back in, it is much more evident that this dataset is actually two linear series with some gaussian noise added. The lines have slightly different slopes and different intercepts, and the noise has different standard deviations. The lines have been plotted here to show you this is most definitely a linear dataset by design albeit a bit noisy. This would be a good candidate for linear regression. Despite there being two distinct linear series, let's first look at the results of a one dimensional linear regression predicting y from x to start building an intuition. Then you'll see if you can do better!

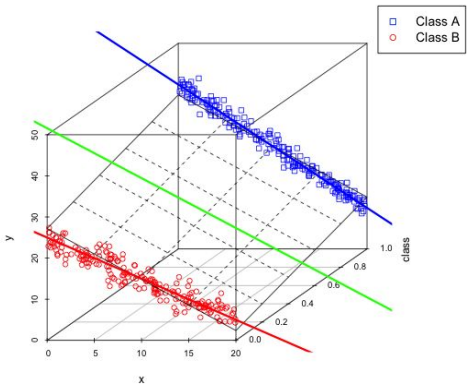
The green line here is the fitted linear equation from linear regression. Notice that it is far away from each individual class' distribution because class B pulls the line away from class A and vice versa. It ends up approximately bisecting the space between the two distributions. This makes sense since with regression we optimize our loss of mean squared error so with an equal pull from each class the linear regression should have the lowest mean squared error in between the two classes approximately equidistant from their means.

Since each class is a different linear series with different slopes and intercepts, we would actually have much better accuracy by performing a linear regression for each class which should fit very closely to each of the lines plotted here. Even better, instead of performing a one dimensional linear regression predicting the value of y from one feature, x, we could perform a two dimensional linear regression predicting y from two features: x and the class of the point. The class feature could be a 1 if the point belongs to class A and a 0 if the point belongs to class B. Instead of a line, it

would form a 2D hyperplane.

Let’s see how that would look.

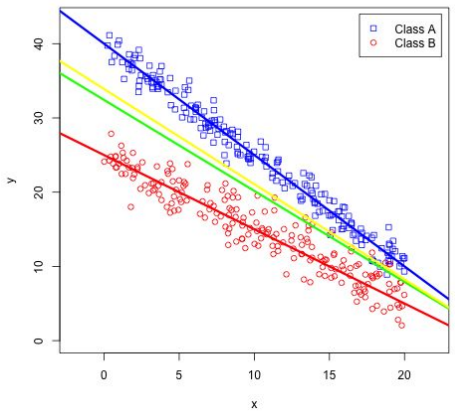
Is this dataset a good candidate for linear regression and/or linear classification?



Here are the results of the 2D linear regression. To predict our label y, we used two features: x and class. As you can see a 2D hyperplane has been formed between the two sets of data, which are now separated by the class dimension. I've also included the true lines for both class A and class B as well as the 1D linear regression's line of best fit. The plane doesn't completely contain any of the lines, due to the noise of the data tilting the two slopes of the plane. Otherwise with no noise, all three lines would be perfectly on the plane.

Also, we have kind of already answered the other portion of the quiz question about linear classification, because the linear regression line does a really great job already of separating the classes. So this looks like a very good candidate for linear classification as well. But would it produce a decision boundary exactly on the 1D linear regression's line of best fit? Let's find out!

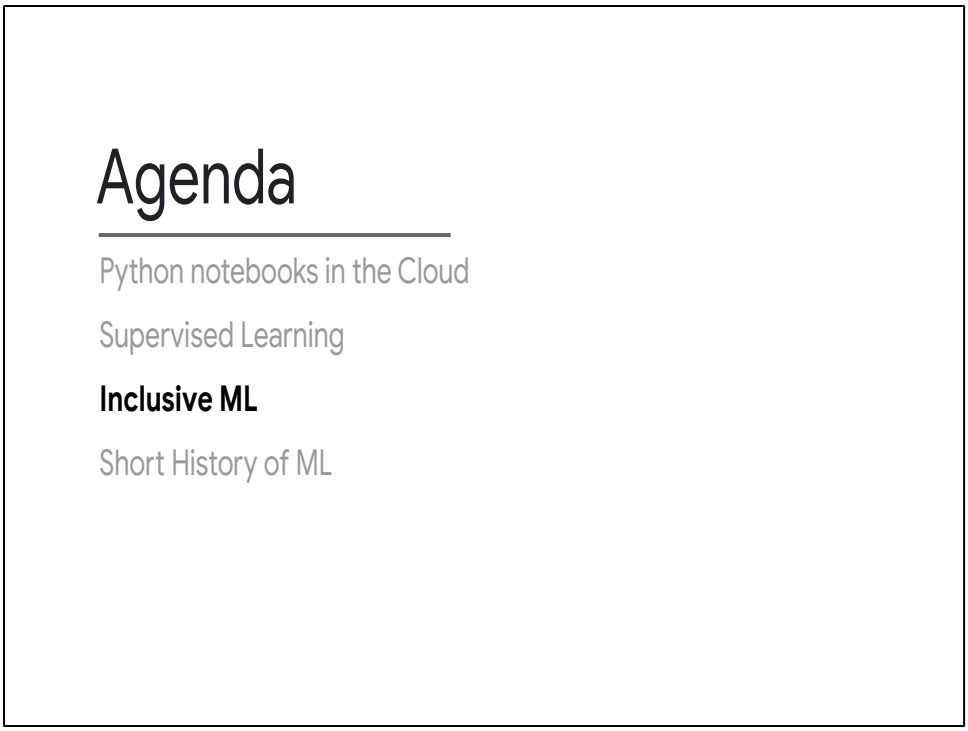
Is this dataset a good candidate for linear regression and/or linear classification?



Plotted in yellow is the output of a one dimensional linear classifier, logistic regression. Notice that it is very close to the linear regression's green line but not exactly. Why could this be? Remember, I mentioned that regression models usually use mean squared error as their loss function whereas classification models tend to use cross entropy. So what is the difference between the two? Without going into the details too much just yet, there is a quadratic penalty for mean squared error so it is essentially trying to minimize the euclidean distance between the actual label and predicted label.

On the other hand, with classification's cross entropy the penalty is almost linear looking when the predicted probability is close to the actual label but as it gets farther away it becomes exponential when it gets close to predicting the opposite class of the label. Therefore, if you look closely at the plot, the most likely reason the classification decision boundary line has a slightly more negative slope is so that some of those noisy red points, red being the noiser distribution, fall on the other side of the decision boundary and lose their high error contribution. Since they are close to the line their error contribution would be small for linear regression because not only is it the error only quadratic but there is no preference to be on one side of the line or the other for regression, as long as the distance is as small as possible.

So as you can see, this dataset is a great fit for both linear regression and linear classification, unlike when we looked at the tips dataset where it was only acceptable for linear regression and would be better for nonlinear classification.

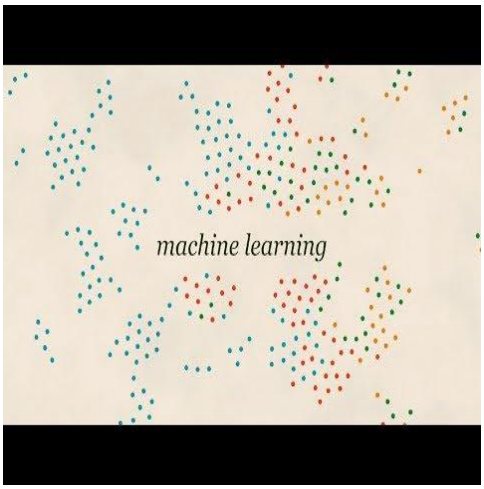


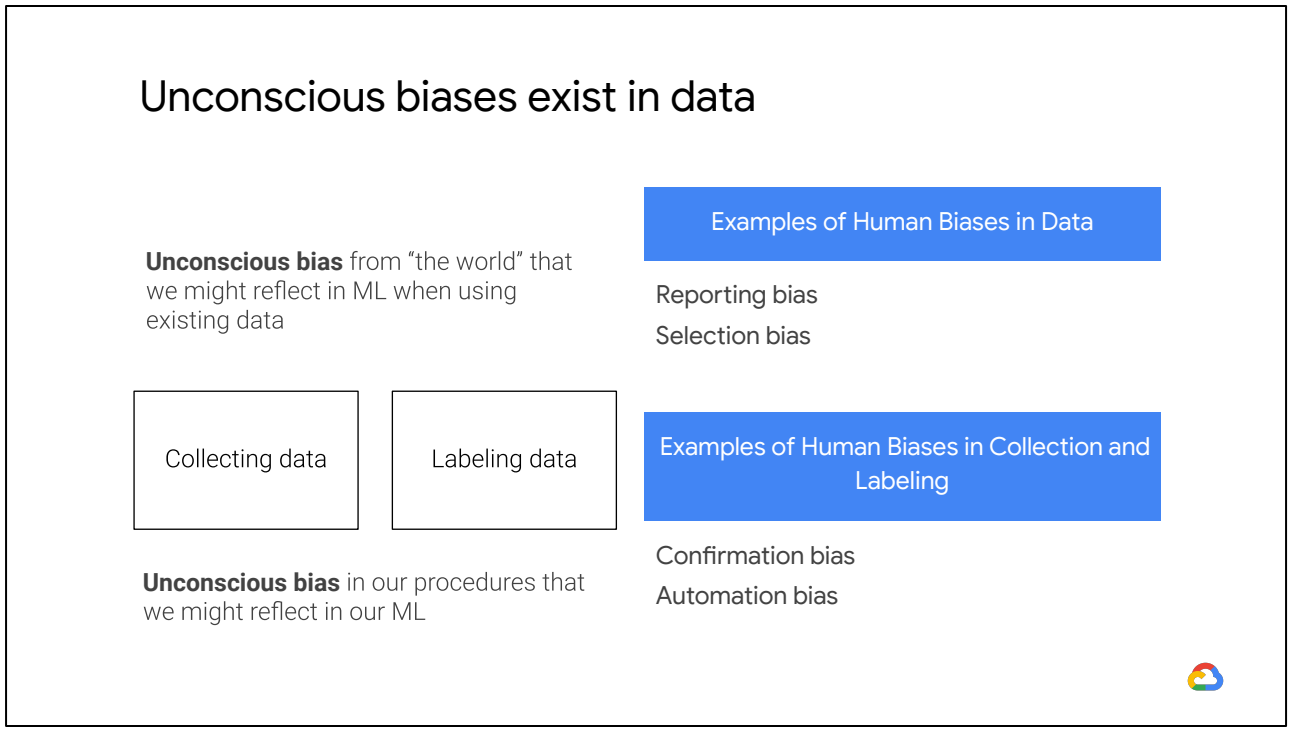
So far, we've talked about ML strategy -- about what machine learning means, what problems it can solve, and how to put it into practice at your company. Besides these technical and business aspects, another aspect that you want to keep in mind is that whatever ML models you build have to treat all your customers fairly. A key aspect of your ML strategy has to be ... building machine learning systems in an inclusive way.

In this module, I will show you how to identify the origins of bias in machine learning -- it comes down to the training data. Then, I will show you ways in which you can apply an inclusive lens throughout the machine learning development process—from the data exploration, all the way over to evaluating the performance of your trained model.
We will also discuss practical techniques to make your models inclusive.

Let's delve in.

Human biases lead to biases in ML models





Unconscious biases exist in our data. They exist in two forms:

First, there are the human biases that exist in data because data found in "the world" has existing biases with regard to properties like gender, race, and sexual orientation. We can think of those biases in terms of the way they manifest in our data samples.

There may be *reporting bias* by our subjects because they only choose to reveal certain aspects about themselves or their opinions. There may be *selection bias*; that is, those subjects that get into our samples only represent a privileged type of user.

There may be *selection bias*; that is, those subjects that get into our samples only represent a privileged type of user.

Let's say that we are developing an ML model to find fraudulent transactions. Can you see what kinds of problems we might have?
Suppose we train the model on historical transactions. What could go wrong there? Well … we may have a reporting bias if our customers tend to use cash in grocery stores. We only know about the places where they tend to shop with the card, and we might wrongly assume that all grocery store transactions are fraudulent. Just because we have not seen enough grocery store transactions that use a card. However, many US states issue state benefits in the form of debit cards and this will have the impact of denying poor users the ability to use those cards for food. This is an example of reporting bias that caused a real-world outcome.

Or …

Maybe our cards get used at only large department stores. Our model might start to reject transactions at smaller stores because there aren't enough of those. That's an example of selection bias from the perspective of the small business owner who doesn't get any business from the credit cards we are issuing.

Second, we can also run into human biases which arise as part of our data collection and labeling procedures.

Confirmation bias refer to only looking for data which may confirm our hypotheses.

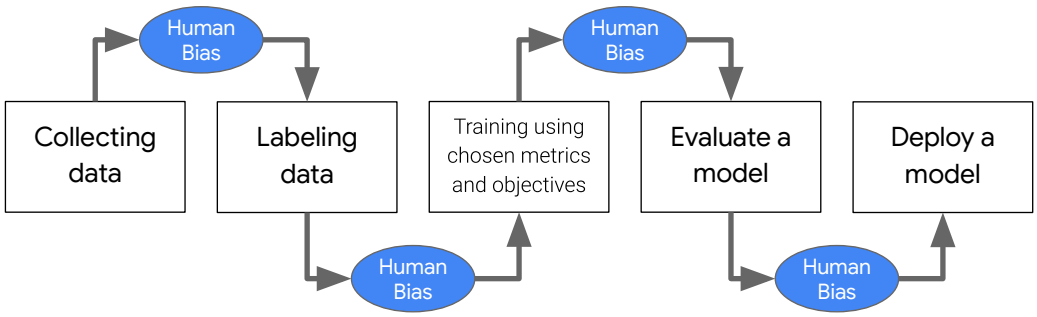
And *automation bias* refers to the biases which crop up when the data we use is solely the data which is easily automatable.

For example, suppose we are building a product recommendation system.
We believe that someone who buys a tie will look for men's shirts. So, we might restrict our data collection to only men's apparel items.
But lots of women do wear ties. This is an example of confirmation bias.

Automation bias is surprisingly common. If men's apparel and women's apparel are run by different parts of our organization, it is quite possible that this is why we might have collected only men's shirt sales. The presence of a silo has caused unconscious bias.
You might also have automation bias if you have a fancy digital data collection system at some locations, but other locations are still sending you faxes.

Chances are that you will prioritize the data that is easy to collect.

A typical ML pipeline *with bias*



So, what's the impact of biases in collecting data and labeling data?
It affects the entire pipeline!

In machine learning we begin with our data; the biases in the original data are going to be reflected downstream in our models and consequently are going to result in potentially biased outcomes.

The result is that biases can appear in at every point of the data pipeline. It appears at the point of collection. What kinds of data are accessible? Does the availability of data privilege a particular group of people? Does it put another group at a disadvantage? What transformations are you making to your data that may exclude one group compared to another?

It can also appear in the process of labelling. What human biases are human annotators introducing into the data? How stable are your categories of classification?

It can occur at the model level. Does a particular objective put certain subgroups of people at a disadvantage compared to the group in aggregate?

And those biases will appear downstream in the output of our model, and our users will see an effect or outcome as a result.

2 Avoid creating or reinforcing unfair bias

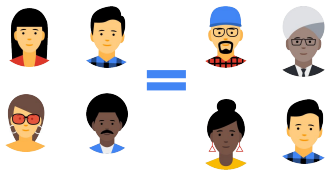
ML models learn from existing data collected from the real world, and so an accurate model may learn or even amplify problematic pre-existing biases in the data based on race, gender, religion, or other characteristics.

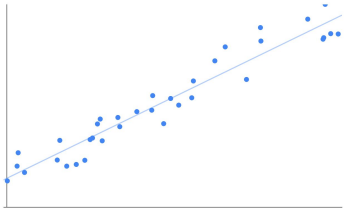
ai.google/principles

We recognize that such powerful technology raises equally powerful questions about its use. How AI is developed and used will have a significant impact on society for many years to come. As a leader in AI, we feel a deep responsibility to get this right. So Google has announced seven AI principles to guide our work going forward. These are not theoretical concepts; they are concrete standards that will actively govern our research and product development and will impact our business decisions.

The challenge we run into when creating a system that is fair and inclusive to all is that ML models learn from *existing* data collected from the real world, so an accurate model may learn or even amplify problematic pre-existing biases in the data based on race, gender, religion, or other characteristics.

A Checklist for Bias-Related Issues





Here's a quick checklist for situations where you should watch out for bias-related issues.

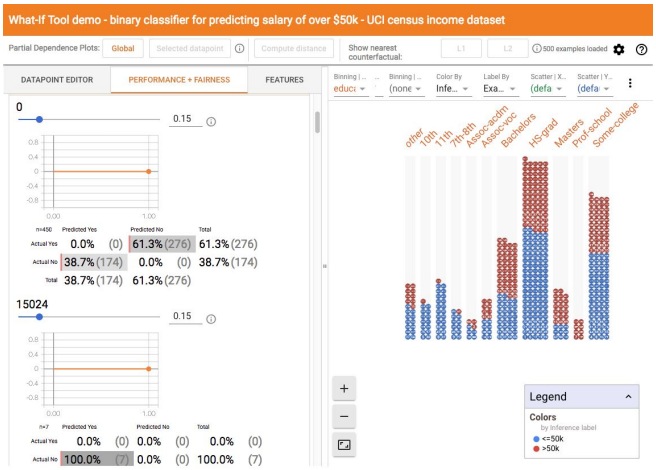
Does your use case or product specifically use any of the following data: biometrics, race, skin color, religion, sexual orientation, socioeconomic status, income, country, location, health, language, or dialect?

Does your use case or product use data that is likely to be highly correlated with any of the personal characteristics listed above (for example, zip code or other geospatial data is often correlated with socioeconomic status and/or income; image/video data can reveal information about race, gender, and age)?

Could your use case or product negatively affect individuals' economic or other important life opportunities?

https://cloud.google.com/inclusive-ml/#fairness-in-ml-automl

Tools for Responsible AI



There are tools available from Google that help you diagnose fairness issues in your data, in your labels, and in the effects of predictions.
If you are implementing models, please use them.

This is the What-If tool. It's available for free and you will be able to access it from within Tensorboard. It's designed to let you: visualize inference results, Edit a datapoint and see how your model performs, Explore the effects of a single feature, Arrange examples by similarity, View confusion matrices and other metrics and Test algorithmic fairness constraints.

So far, we've talked about ML strategy -- about what machine learning means, what problems it can solve, and how to put it into practice at your company. Besides these technical and business aspects, another aspect that you want to keep in mind is that whatever ML models you build have to treat all your customers fairly. A key aspect of your ML strategy has to be ... building machine learning systems in an inclusive way.

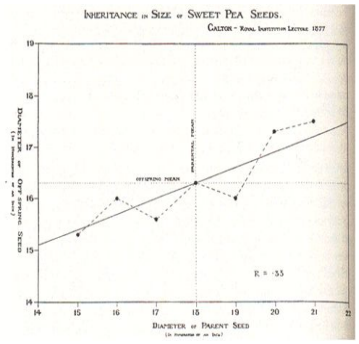
In this module, I will show you how to identify the origins of bias in machine learning -- it comes down to the training data. Then, I will show you ways in which you can apply an inclusive lens throughout the machine learning development process—from the data exploration, all the way over to evaluating the performance of your trained model.
We will also discuss practical techniques to make your models inclusive.

Let's delve in.

Linear regression was invented when computations were done by hand, but it continues to work well for large datasets

Linear Regression
For predicting planets and pea growth

1800s



Linear regression was “invented” for predicting the movement of planets and the size of pea pods based on their parents.
Sir Francis Galton pioneered the use of statistical methods to measurements of natural phenomena.
He was looking at data on the relative sizes of parents and their children in various species, including sweet peas.
He observed something that is not immediately obvious ... something rather strange.
Sure, a larger-than-average parent tends to produce a larger-than-average child.
But how much larger is that child to the average of other children in its generation?
It turned out that this ratio for the child is *less* than the corresponding ratio for the parent.

For example, if the parent's size is 1.5 standard deviations from the mean within its own generation, then you would predict that the child's size will be less than 1.5 standard deviations from the mean within its cohort.

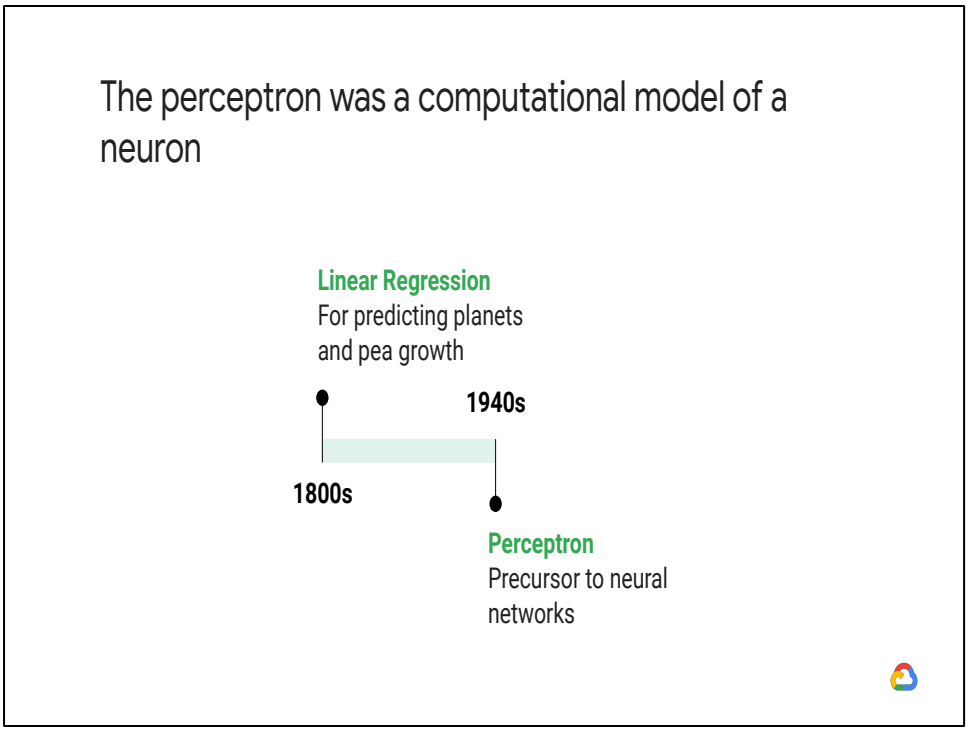
We say that generation-by-generation, things in nature regress, or go back, to the mean. Hence the name “linear regression”. This chart here, from 1877, is the first ever linear regression. Pretty cool!

Compute power in the 1800s was somewhat limited, so they didn’t even realize how well this would continue to work once we had large datasets.

There is actually a closed form solution for solving linear regression, but gradient

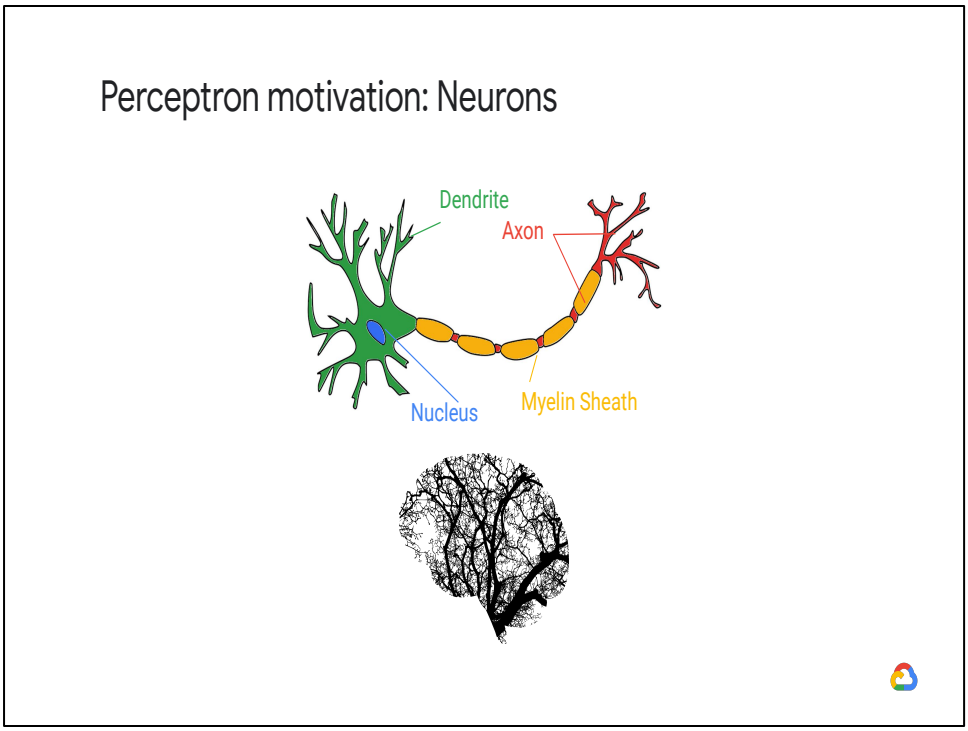
descent methods can also be used, each with their pros and cons, depending on your dataset. Let's look under the hood on how linear regression works.

Image from http://people.duke.edu/~rnau/regintro.htm, but is of book that is out of copyright.



So, linear regression was pretty much it as far as learning from data was concerned until the 1940s ...

A researcher, Frank Rosenblatt, comes up with the perceptron as a computational model of a neuron in the human brain and shows how it can “learn” simple functions. It is what we would call today a binary linear classifier, where we are trying to find a single line that splits the data into two classes. A single layer of perceptrons would be the simplest possible feedforward neural network. Inputs would feed into a single layer of perceptrons, and a weighted sum would be performed. This sum would then pass through what we call today an activation function, which is just a mathematical function you apply to each element that is now residing within that neuron. Remember though at this point, this is still just a linear classifier, so the activation function, which is linear, in this case just returns its inputs. Comparing the output of this to a threshold would then determine which class each point belongs to. The errors would be aggregated and used to change the weights used in the sum, and the process would happen again until convergence.



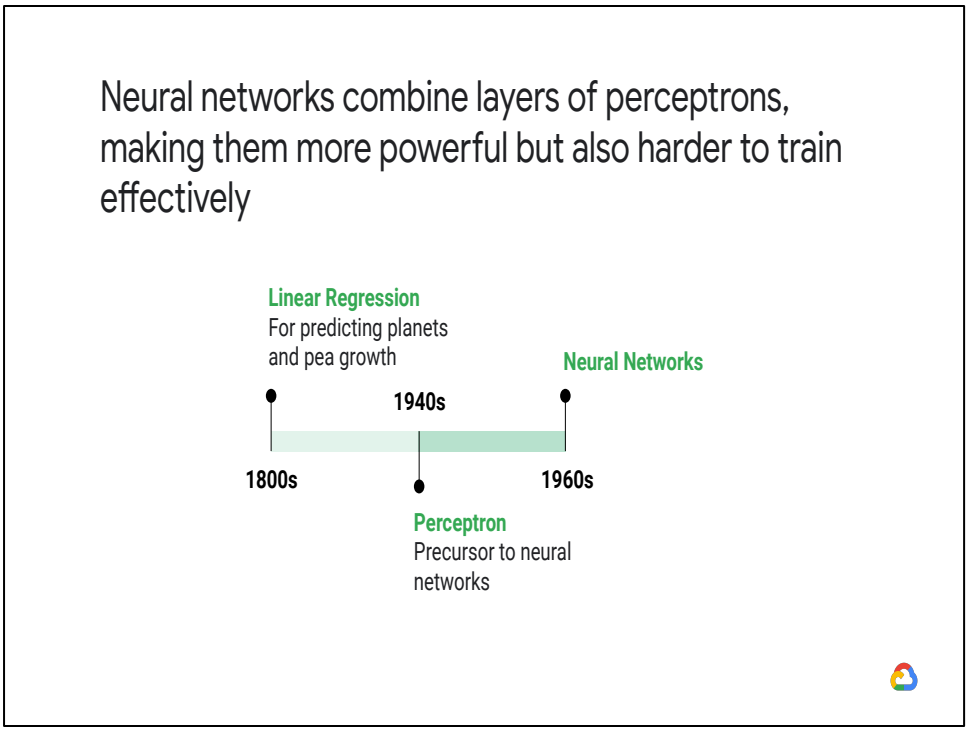
If you are trying to come up with a simple model of something that learns a desired output from a given input distribution then you needn't look far since our brains do this all day long making sense out of the world around us and all of the signals our bodies receive. One of the fundamental units of the brain is the neuron. Neural networks are just groups of neurons connected together in different patterns or architectures. A biological neuron has several components specialized in passing along electrical signal which allows you and I to have thoughts, perform actions, and study the fascinating world of machine learning.

Electrical signals from other neurons, such as sensory neurons in the retina of your eye, are propagated from neuron to neuron. The input signal is received at one end of the neuron which is made up of dendrites. These dendrites might not just collect electrical signal from just one other neuron but possibly from several which all get summed together over windows in time that changes the electrical potential of the cell. A typical neuron has a resting electric potential of -70mV. As the input stimuli received at the dendrites increases, eventually it reaches a threshold around -55mV in which case a rapid depolarization of the axon occurs with a bunch of voltage gates opening up allowing a sudden flow of ions. This causes the neuron to fire an action potential of electric current along the axon, aided by the myelin sheath for better transmission to the axon terminals. Here neurotransmitters are released at synapses that then travel across the synaptic cleft to usually the dendrites of other neurons. Some of the neurotransmitters are excitatory where they raise the potential of the next cell while some are inhibitory and lower the potential. The neuron repolarizes to an even lower potential than resting for a refractory period and then the process

continues at the next neuron until maybe it eventually reaches a motor neuron and moves your hand to shield the sun out of your eyes.

So what does all of this biology and neuroscience have to do with machine learning?

Graphic: Ryan, Modified to make a Google Themed Neuron:
https://pixabay.com/en/neuron-nerve-cell-axon-dendrite-296581/ cc0
Graphic: (human brain) cc0:
https://pixabay.com/en/brain-anatomy-abstract-art-2146817/

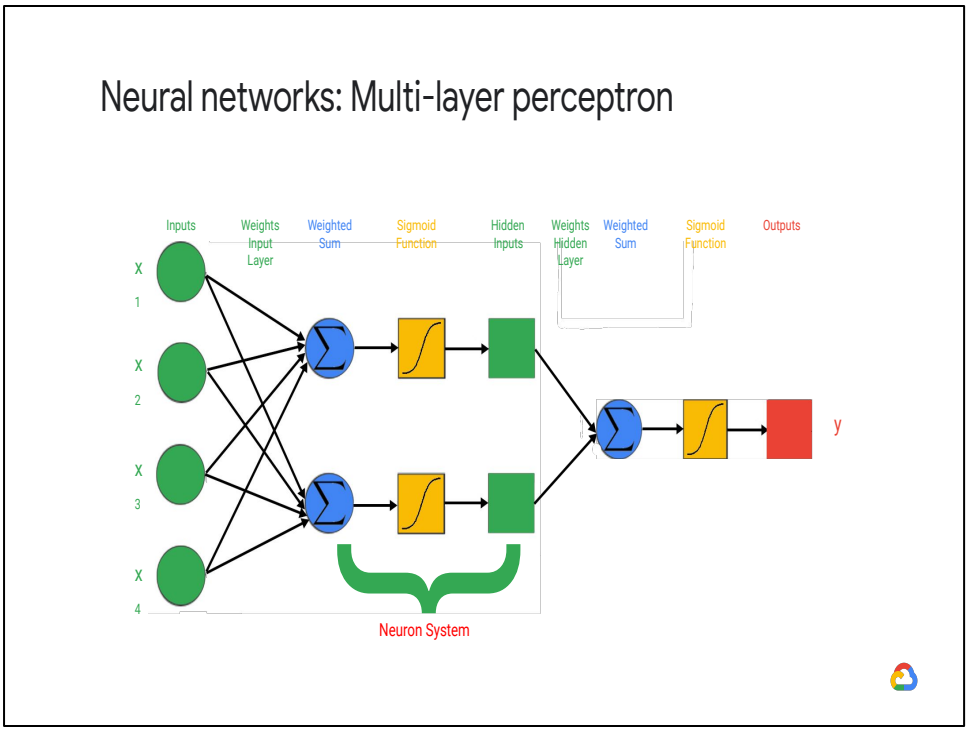


Why only one layer of perceptrons? Why not send the output of one layer as the input to the next layer?
Combining multiple layers of perceptrons sounds like it would be a much more powerful model.

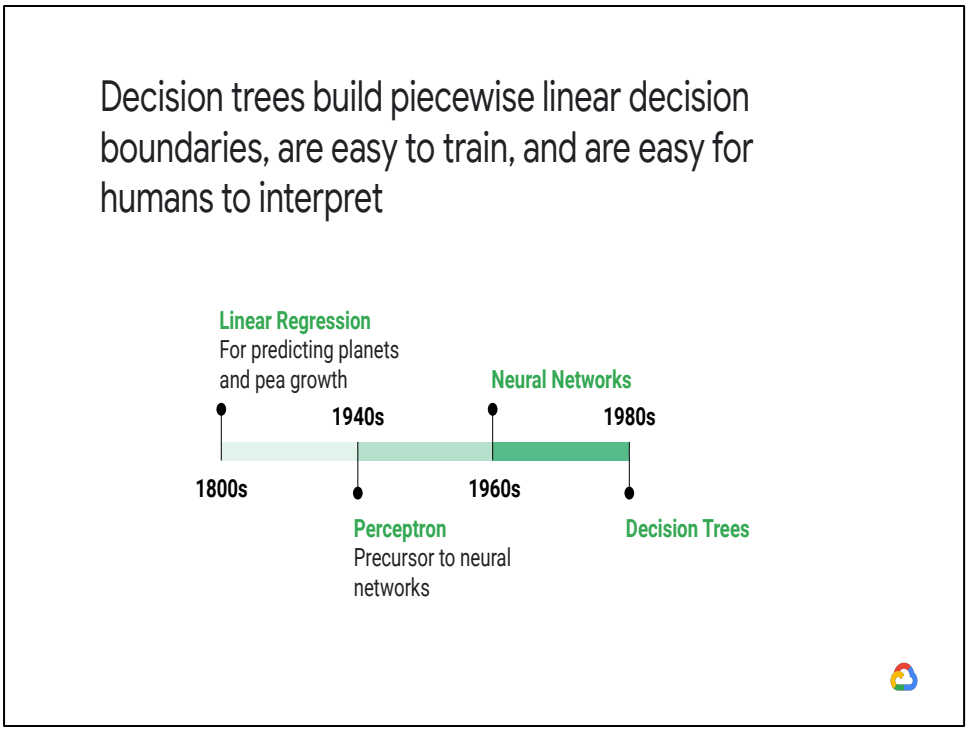
However without using nonlinear activation functions, all of the additional layers can be compressed back down into just a single linear layer, and there is no real benefit. You need nonlinear activation functions.
Therefore, sigmoid or tanh activation functions started to be used for nonlinearity. At the time, we were limited to just these because we needed a differentiable function since that fact is exploited in backpropagation to update the model weights.

Modern-day activation functions are not differentiable, and people didn't know how to work with them.
This constraint -- that activation functions had to be differentiable -- could make the networks hard to train.
The effectiveness of these models was also constrained by the amount of data, available computational resources, and other difficulties in training.
For instance, optimization tended to get caught in saddle points instead of finding the global minimum we hoped it would during gradient descent.

However, once the trick to use Rectified Linear Units, or ReLUs, was developed, then you could have faster training like 8 to 10X; almost guaranteed convergence for logistic regression.

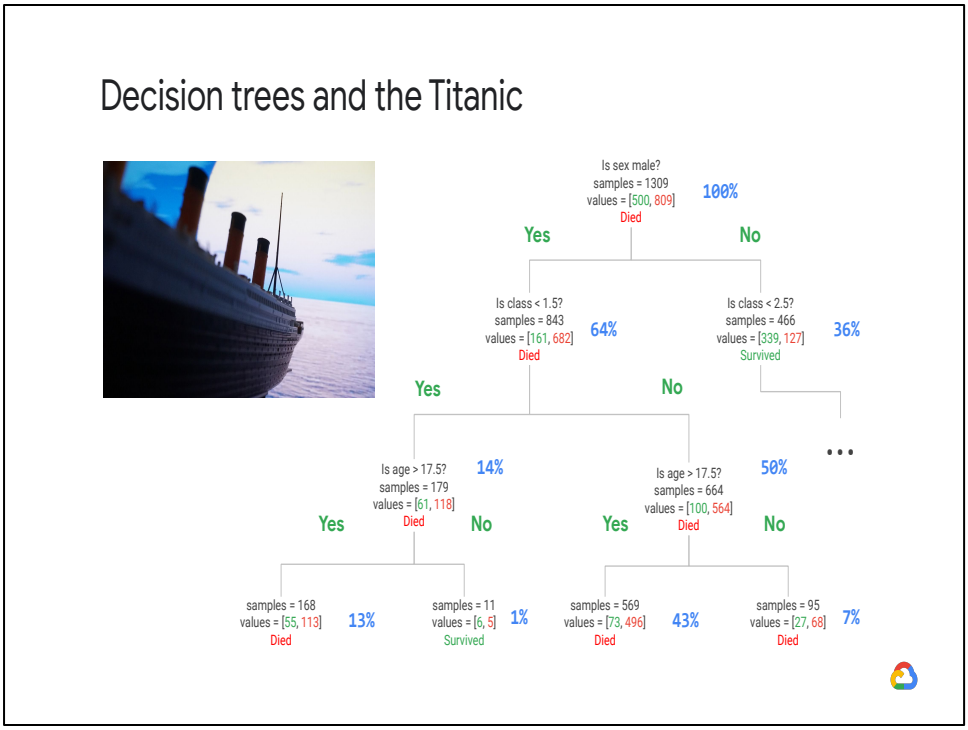


Building off of the perceptron, just like the brain, we can connect many of them together to form layers to create feedforward neural networks. Really not much has changed in components from the single layer perceptron. There are still inputs, weighted sums, activation functions, and outputs. One difference is that the inputs to neurons not in the input layer are not the raw inputs but the outputs of the previous layer. Another difference is that the weights connecting the neurons between layers are no longer a vector but now a matrix because of the completely connected nature of all neurons between layers. For instance, in the diagram the input layer weights matrix is 4 x 2 and the hidden layer weights matrix is 2 x 1. We will learn later that neural networks don't always have complete connectivity which has some amazing applications and performance like with images. Also, there are different activation functions than just the unit step function such as the sigmoid and the hyperbolic tangent or tanh activation functions. Each non-input neuron, you can think of as a collection of three steps packaged up into a single unit. The first component is a weighted sum, the second component is the activation function, and the third component is the output of the activation function.



Decision tree algorithms such as ID3 and C4.5 are invented in the 80s/90s. They are better at certain types of problems than linear regression and are very easy for humans to interpret. Finding the optimal splitting when creating the trees is an NP-hard problem, therefore greedy algorithms were used to hopefully construct trees as close to optimal as possible.

They create a piecewise linear decision surface which is essentially what a layer of ReLUs gives you. But with DNNs or Deep Neural Networks, each of the ReLU layers combines to make a hyper planar decision surface, which can be much more powerful. But I am skipping ahead to why DNNs can be better than decision trees. Let's first talk about decision trees



Decision trees are one of the most intuitive machine learning algorithms. They can be used for both classification and regression. Imagine you have a dataset and you want to determine how the data is all split into different buckets. The first thing you should do is brainstorm some interesting questions to query the dataset with. Let's walk through an example. Here is the well known problem of predicting who lived or died in the Titanic catastrophe. There were people aboard from all walks of life, different backgrounds, different situations, etc. so we want to see if any of those possible features can partition my data in such a way that I can with high accuracy predict who lived.

A first guess at a feature could possibly be the sex of the passenger. Therefore I could ask the question, is the sex male? Thus I split the data with males going into one bucket and the rest going into another bucket. 64% of the data went into the male bucket leaving 36% going into the other one. Let's continue along the male bucket partition for now.

Another question I could ask is about what passenger class each passenger was. With our partitioning, now 14% of all passengers were male and of the lowest class whereas 50% of all passengers were male and of the higher two classes. The same type of partitioning could also continue in the female branch of the tree. Taking a step back, it is one thing for the decision tree building algorithm to split sex into two branches because there are only two possible values, but how did it decide to split passenger class with one passenger class branching to the left and two passenger classes branching to the right?

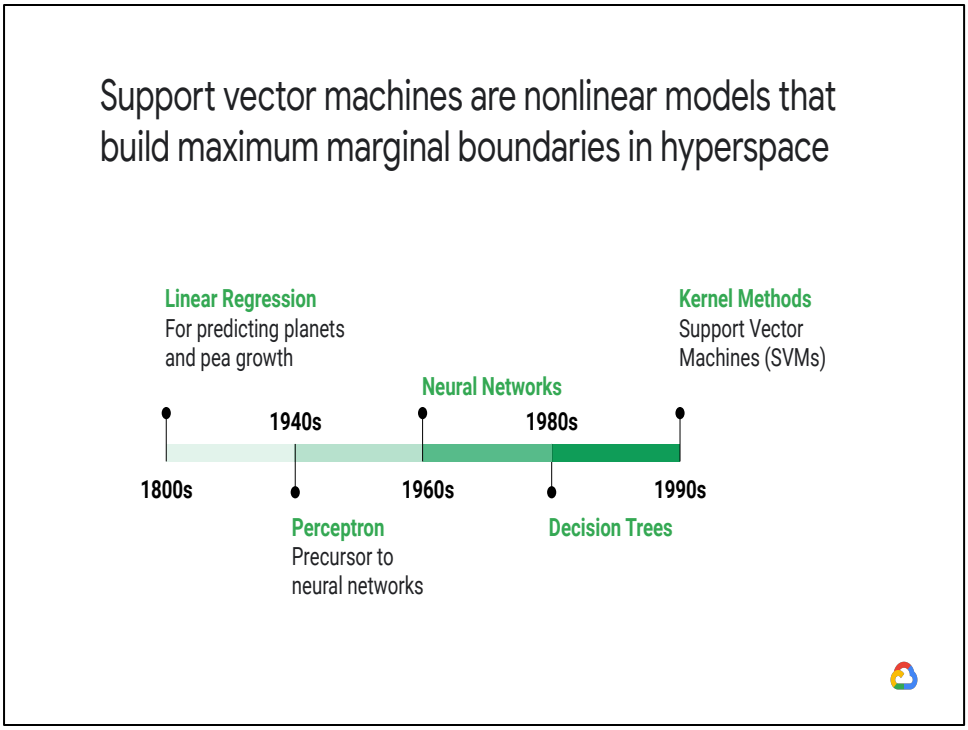
For instance, in the simple Classification And Regression Tree or CART algorithm, the algorithm tries to choose a feature and threshold pair that will produce the purest subsets when split. For classification trees, a common metric to use is the Gini impurity, but there is also entropy. Once it has found a good split, it searches for another feature threshold pair and splits that into subsets as well. This process continues on recursively until either the set maximum depth of the tree has been reached or if there are no more splits that reduce impurity. For regression trees, mean squared error is a common metric to split on.

Does this sound familiar how it chooses to split the data into two subsets? Each split is essentially just a binary linear classifier that finds a hyperplane that slices along one feature's dimension at some value which is the chosen threshold to minimize the members of a class falling on the other class' side of the hyperplane. Recursively creating these hyperplanes in a tree is analogous to layers of linear classifer nodes in a neural network. Very interesting! Now that we know how decision trees are built, let's continue building this tree a bit more.

Perhaps there is an age threshold that will help me split my data well for this classification problem. I could ask is the age greater than 17.5 years old? Looking at the lowest class branch of the male parent branch, now just 13% of the passengers were 18 and older while only 1% were younger. Looking at the classes associated with each node, only this one on the male branch so far is classified as survived. We could extend our depth and/or choose different features to hopefully keep expanding the tree until every node only has passengers that had survived or died.

However, there are problems with this because essentially I am just memorizing my data and fitting the tree perfectly to it. In practice, we are going to want to generalize this to new data and a model that has memorized the training set is probably not going to perform very well outside of it. There are some methods to regularize it, such as setting a minimum number of samples per leaf node, a maximum of leaf nodes, or a maximum number of features. You can also build the full tree and then prune unnecessary nodes.

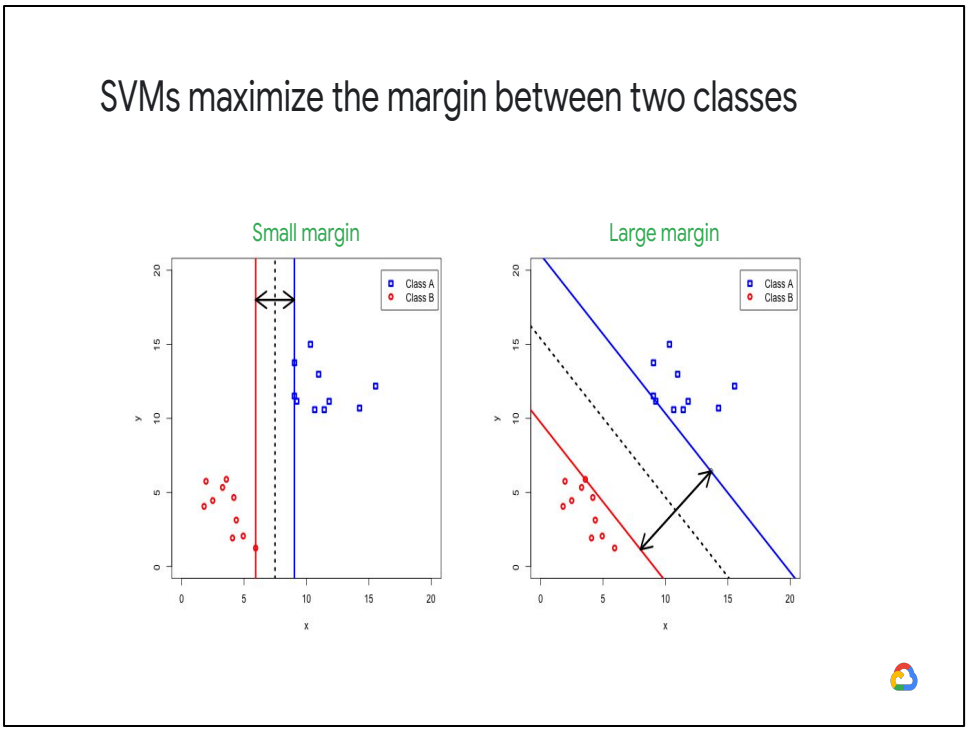
To really get the most out of trees it is usually best to combine them into forests which we will talk about very soon.



Starting in the 1990s, the field of kernel methods was formed. Corinna Cortes, Director of Google Research, was one of the pioneers.

This field of study allows interesting classes of new non-linear models, most prominently nonlinear SVMs or Support Vector Machines, which are maximum margin classifiers that you may have heard of before.

Fundamentally core to an SVM is a nonlinear activation plus a sigmoid output for maximum margins.

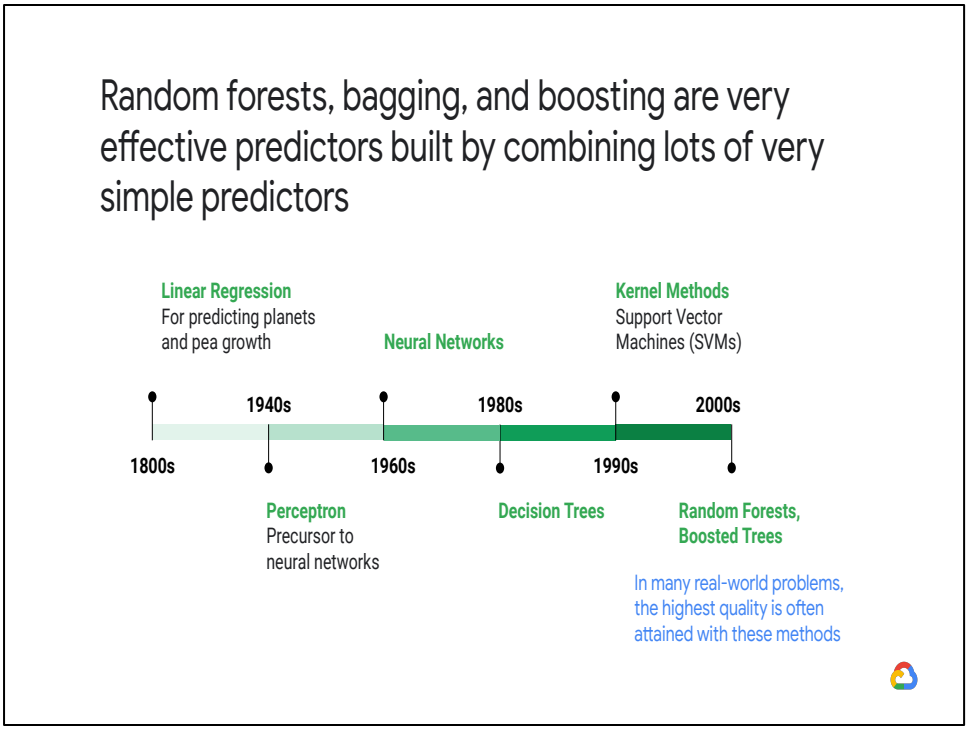


Earlier we have seen how logistic regression is used to create a decision boundary to maximize the log likelihood of the classification probabilities. In the case of a linear decision boundary, logistic regression wants to have each point of the associated class as far from the hyperplane as possible and provides a probability which can be interpreted as prediction confidence. There are an infinite number of hyperplanes you can create between two linearly separable classes such as the two hyperplanes shown as the dotted lines in the two figures here. In SVMs, we include two parallel hyperplanes on either side of the decision boundary hyperplane where they intersect with the closest data point on each side of the hyperplane. These are the support vectors. The distance between the two support vectors is the margin.

On the left we have a vertical hyperplane that indeed separates the two classes, however the margin between the two support vectors is small. By choosing a different hyperplane such as the one on the right, there is a much larger margin. The wider the margin, the more generalizable the decision boundary is which should lead to better performance on new data. Therefore, SVM classifiers aim to maximize the margin between the two support vectors using a hinge loss function compared to logistic regression's minimization of cross entropy.

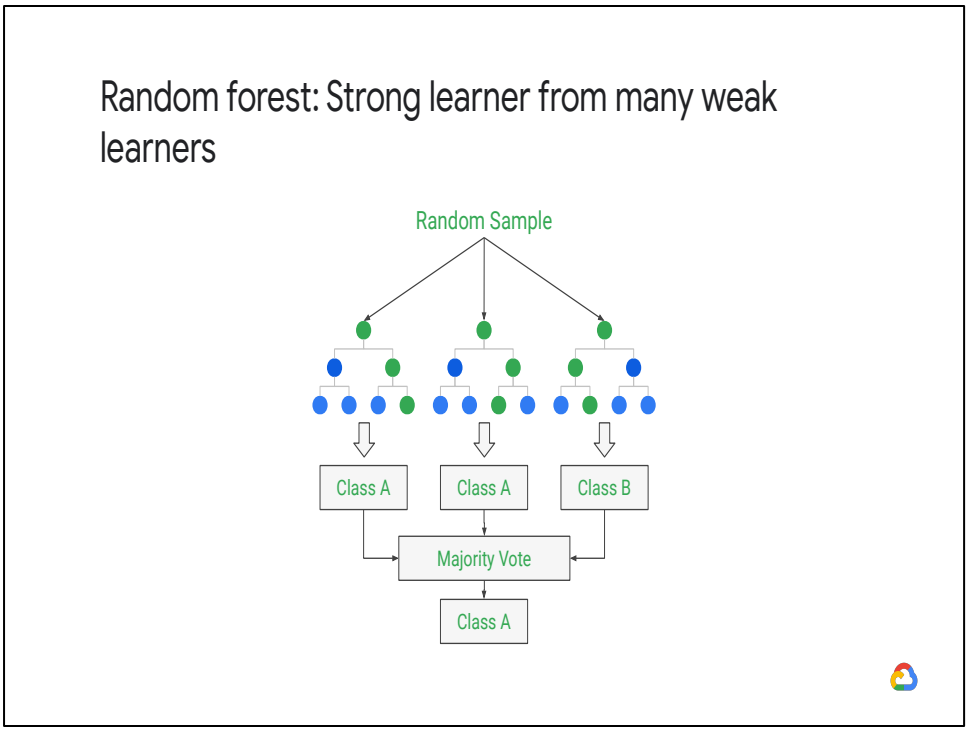
You might notice that I have only two classes which makes this is a binary classification problem. One class' label is given a value of 1 and the other class' label is given a value of -1. If there are more than two classes, then a one versus all approach should be taken and then choose the best out of the permuted binary classifications.

But what happens if the data is not linearly separable into the two classes?



Coming into the last few decades in the 2000s, machine learning research now had the computational power to combine and blend the performance across many models in what we call an ensemble method. You can imagine that, if the errors are independent for a number of simple weak learners, combined they would form a strong learner. DNNs can approximate this by using dropout layers, which help regularize the model and prevent overfitting. This can be simulated by randomly turning off neurons in the network with some probability for each forward pass, which will essentially be creating a “new” network each time.

In many of today’s real-world problems, the highest quality is often attained with these methods.



Oftentimes complex questions are better answered when aggregated from thousands of people's responses instead of those just by a sole individual. This is known as the wisdom of the crowd. The same applies to machine learning when you aggregate the results of many predictors, either classifiers or regressors. The group will usually perform better than the best individual model. This group of predictors is an ensemble which when combined in this way leads to ensemble learning. The algorithm that performs this learning is an ensemble method.

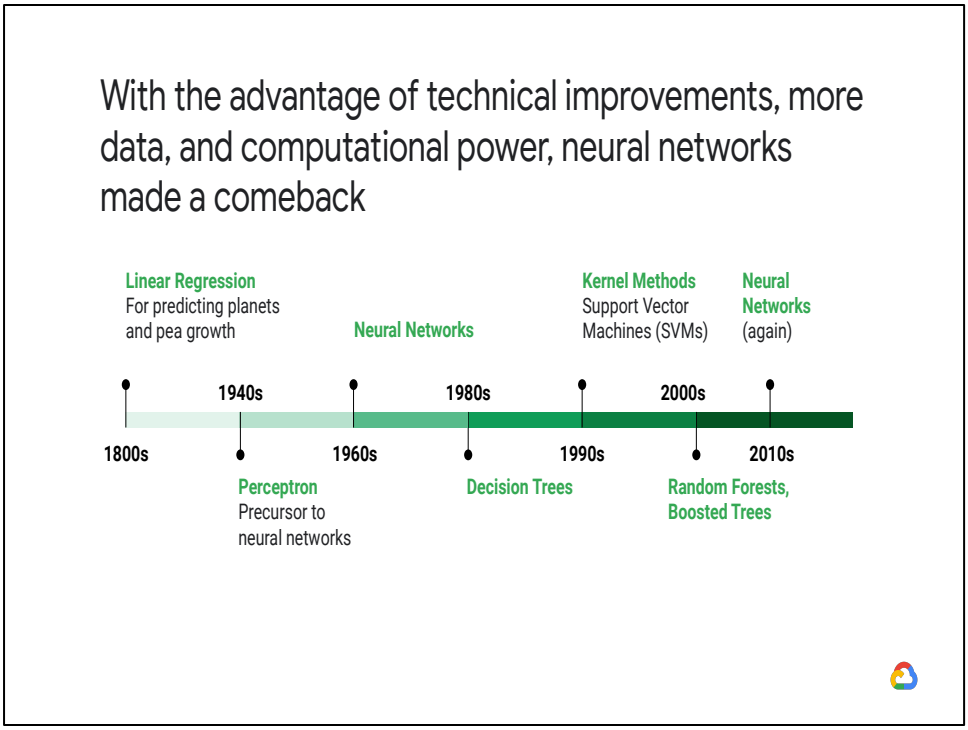
One of the most popular types of ensemble learning is the random forest. Instead of taking your entire training set and using that to build one decision tree, you could have a group of decision trees that each get a random subsample of the training data. Since they haven't seen the entire training set they can't have memorized the whole thing. Once all of the trees are trained on their subset of the data, you can now make the most important and valuable part of machine learning. Predictions! To do so, you would pass your test sample through each tree in the forest and then aggregate the results. If this is classification, there could be a majority vote across all trees which then would be the final output class. If it is regression, it could be an aggregate of the values such as the mean, max, median, etc.

To improve generalization, you can random sample the examples, and/or, the features. We call random sampling the examples with replacement, bagging, short for bootstrap aggregating, and pasting when without replacement. Each individual predictor has higher bias being trained on the smaller subset rather than the full dataset but the aggregation reduces both the bias and variance. This usually gives

the ensemble a similar bias as a single predictor on the entire training set, but with a lower variance. A great method of validation for your generalization error is to use your out-of-bag data, instead of having to have a separate set pulled from the dataset before training. It is reminiscent of k-fold validation using random holdouts. Random subspaces are made when we sample from the features, and, if we random sample examples too, it is called random patches.

Adaptive Boosting or AdaBoost and Gradient Boosting are both examples of boosting which is when we aggregate a number of weak learners to create a strong learner. Typically this is done by training each learner sequentially which tries to correct any issues the learner had before it. For boosted trees, as more trees are added to the ensemble the predictions usually improve. So do we continue to add trees ad infinitum? Of course not! You can use your validation set to use early stopping, so that we don't start overfitting our training data because we have added too many trees.

Lastly, just as we saw with neural networks, we can perform stacking where we can have meta learners learn what to do with predictions of the ensemble, which, can in turn also be stacked into meta meta learners and so on. We will see this subcomponent stacking and reuse in deep neural networks shortly.



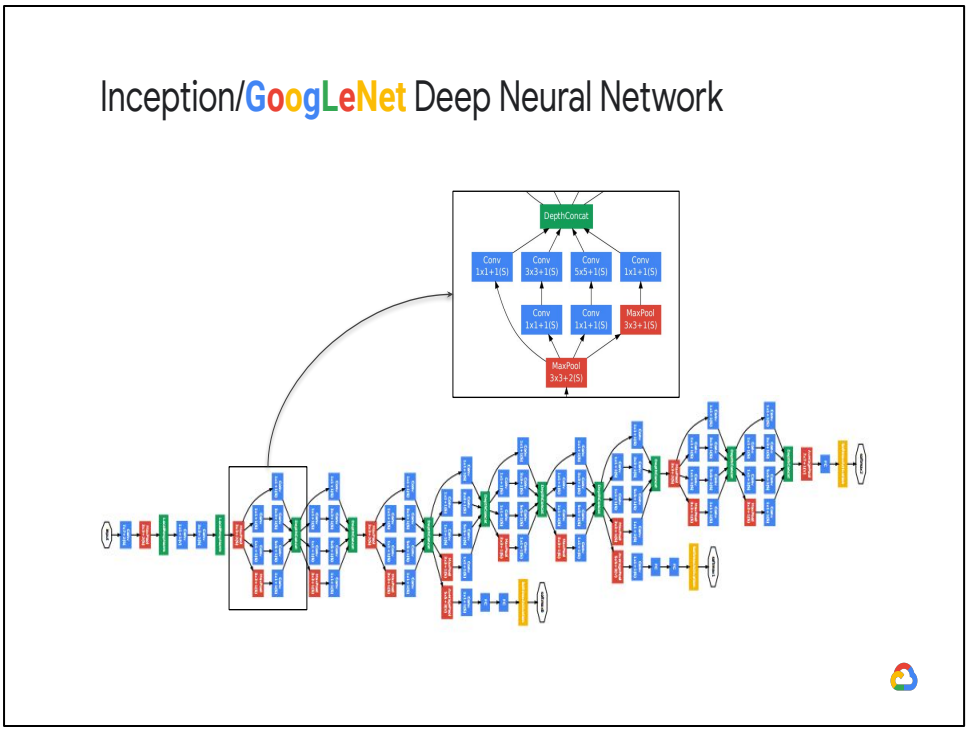
Back again on the timeline are neural networks, now with even more of an advantage through leaps in computational power and lots and lots of data. DNNs began to substantially outperform other methods on tasks such as Computer Vision.

In addition to the boon from boosted hardware, there were many new “tricks” and architectures that helped to improve trainability of deep neural networks like ReLUs, better initialization methods, CNNs or convolutional neural networks, and dropout.

We talked about some of these tricks from other ML models.

The use of non-linear activation functions like ReLUs which usually are set as the default now we talked about during the first look at neural networks.
Dropout layers began being used to help with generalization, which works like ensemble methods which we explored when talking about random forests and boosted trees.
Convolutional layers were added that reduce the computational and memory load due to their non-complete connectedness as well as being able to focus on local aspects for instance images rather than comparing unrelated things in an image.

In other words, all the advances that came about in other ML methods got folded back into neural networks. Let’s look at an example of a deep neural network.

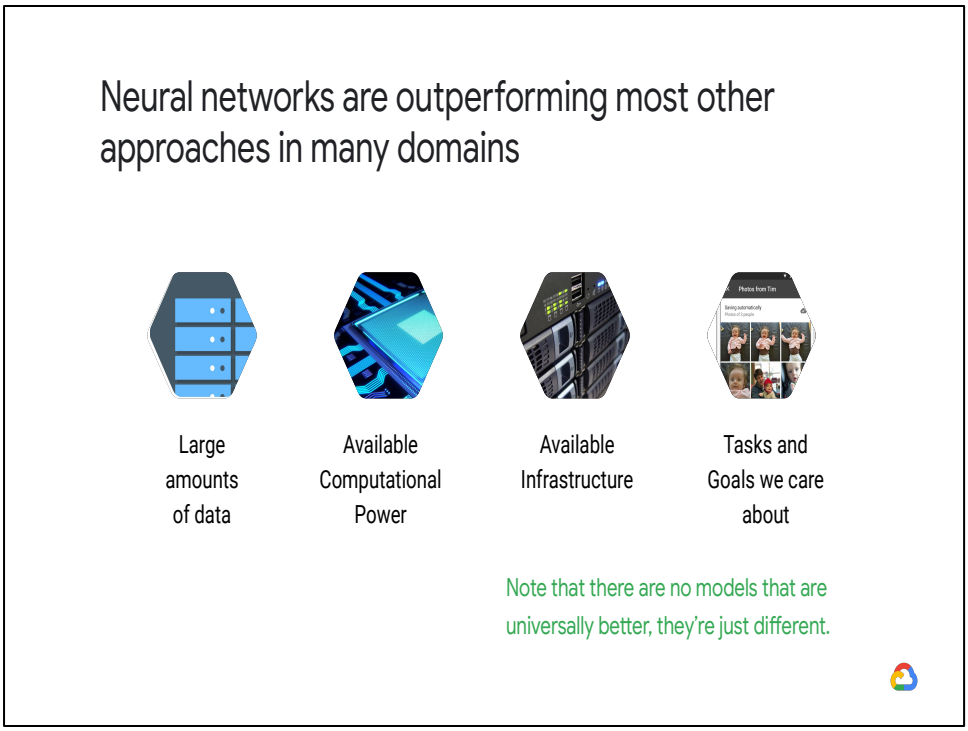


This exciting history of machine learning has culminated into deep learning with neural networks containing 100s of layers and millions of parameters, but with amazing results. Shown here is GoogLeNet or Inception which is an image classification model. It was trained for the ImageNet Large Visual Recognition Challenge in 2014 using data from 2012 where it has to classify images across 1000 classes with 1.2 million images for training. It has 22 deep layers, 27 if you include pooling which we will discuss in a later course, and 100 layers if you break it down into its independent building blocks. There are over 11 million trained parameters.

There are completely connected layers and some that aren't such as convolutional layers which we will talk about later. It used dropout layers to help generalize more, simulating an ensemble of deep neural networks. Just like we saw with neural networks and stacking, each box is a unit of components which is part of a group of boxes such as the one I have zoomed in on. This idea of building blocks adding up to something greater than the sum of its parts is one of the things that has made deep learning so successful. Of course an ever growing abundance of data and faster compute power and more memory helps too. There are now several versions beyond this that are much bigger and have even greater accuracy. The main takeaway from all of this history is that machine learning research reuses bits and pieces of techniques from other algorithms from the past to combine together to make ever powerful models and most importantly, experiment!

Inception: an image recognition model published by Google (From: Going deeper with convolutions, Christian Szegedy et al.)

Graphic: https://cloud.google.com/blog/big-data/2016/07/images/146798944178238/neural-networks-5.png



To recap, the last few decades have seen a proliferation in the adoption and performance of neural networks. With the ubiquity of data, these models have the benefit of more and more training examples to learn from. The increase in data and examples has been coupled with scalable infrastructure for even complex and distributed models with thousands of layers.

One note that we'll leave you with is that although performance with neural networks may be great for some applications, they are just one of the many types of models available for you to experiment with. Experimentation is key to getting the best performance using your data to solve your challenge.