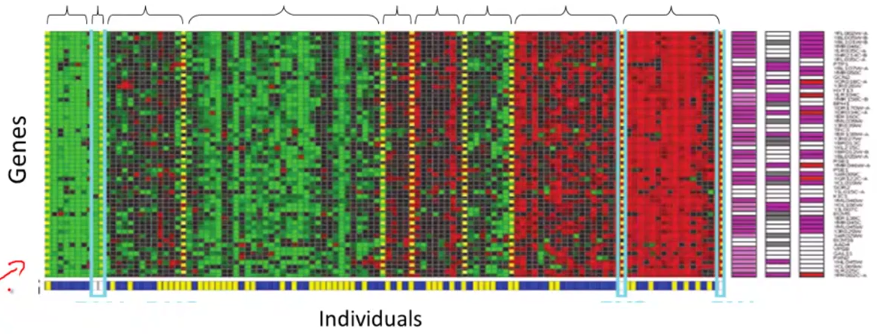
*Stanford Machine Learning – Intro to ML*

* **Machine Learning** - Grew out of work in field AI + is a new capability for CPUs
* Did not know how to program machines to do more complex work (photo tagging, web search, anti-spam filters)
* **Computer vision =** area of CS dealing w/ how CPU’s can perform various visual tasks, imitating behavior of humans (recognizing objects in images, tracking objects in videos, etc.)
* Needed to have machines learn to do them by itself
* Examples: Database mining
* Now have large datasets due to growth of automation/web (Web click/click-stream data, electronic medical records, biology (gene + DNA sequences), engineering, etc.)
* Applications that can’t be programmed by hand
* Autonomous helicopter, handwriting recognition, most of NLP, Computer Vision
* Self-customizing programs
* Amazon product recommendations, understanding human learning (brain + real AI)
* 1959 Definition of **Machine Learning** = Field of study that gives CPUs the ability to learn w/out being explicitly programmed (Arthur Samuel + checkers player)
* 1998 Definition of **Machine Learning** = A program is said to learn from experience **E** w/ respect to some task **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with/after experience **E**
* Ex: email program watches you filter spam, + based on what you do and do not filter, learns how to better filter spam 🡪 T = act of classifying an email as spam or not, E = watching you label the emails, P = % of emails correctly ID’ed as spam
* Ex: Playing Checkers 🡪 T = actually playing, E = experience of playing, P = probability of winning the next game
* Main 2 ML algorithms = Supervised + Unsupervised Learning
* Others: Recommender systems, Reinforced learning
* **Supervised Learning 🡪** given a data set *already know*ing what our correct output (“right answer”) should look like, while also having the idea that there is a relationship between the input + output.
* Supervised learning problems are categorized into **regression** and **classification** problems.
* Regression 🡪 trying to predict results within a *continuous* output, or trying to map input variables to *some continuous function*.
* Statistical process for estimating relationships between variables, such as an output and its features/predictors
* Estimates how a “typical” output value changes when one feature is changed
* Classification 🡪 trying to predict results in a *discrete* output (0 or 1, or multiple categories), or trying to map input variables *into discrete categories*.
* Example 1: Given data about the size of houses on the real estate market, try to predict their price (*Price* as a *function* of *size*)
* Note: a continuous value output (price), so this is a regression problem.
* Could turn this into a classification problem by instead making our output *whether a house sells for more than or less than the asking price.*
* Here we’re classifying houses based on price into 2 discrete categories (> or <)
* Example 2:
* (a) Regression - Given a picture of a person, predict age based on the given picture
* (b) Classification - Given a patient w/ a tumor, predict whether it is malignant or benign.
* If we had an infinity # of **features,** we’d need a neat mathematical trick to deal with them (see **SVM**)
* **Unsupervised learning** allows us to approach problems w/ little/no idea as to what results should look like 🡪 given a data set w/ no labels + are not told what to do with it
* Need to find some structure w/in it
* Can derive structure from data where we don't necessarily know the effect of the variables.
* Can derive this structure by **clustering** data based on relationships among variables in the data.
* *NO feedback based on the prediction results.*
* Example 1: Clustering:
* Take a collection of 1M different genes + find a way to automatically group them into *groups* that are *somehow similar*/related by different variables, such as lifespan, location, roles, + so on.
* Take a collection of *genes being expressed* in individuals (via color gradient) + group individuals into similar groups w/out knowing types of people in advance (no right answer)



* Google News 🡪 looks throughout stories on the web + groups them into cohesive news stories (multiple URL’s in each cluster to display together) for trending stories
* Organize large CPU clusters/data centers to see which machines work better together + put them together to get the CPUs to make the data center work more efficiently
* Social Network analysis 🡪 by knowing who you email/Facebook friends, etc. we can automatically ID cohesive groups of friends that all know each other
* Market Segmentation 🡪 look at customer data + group customers into different market segments to more efficiently market to different segments
* Astronomical Data Analysis 🡪 useful theories on how galaxies are formed
* Example 1: NON-clustering:
* The "**Cocktail Party Algorithm**", allows you to find structure in a chaotic environment. (i.e. identifying individual voices + music from a mesh of sounds at a [cocktail party](https://en.wikipedia.org/wiki/Cocktail_party_effect)).
* Imagine a party in a room full of people all talking at the same time (causes many overlapping voices), so it’s almost hard to hear the person in front of you.
* At a cocktail party w/ 2 people talking at the same time w/ 2 microphones in the room at 2 different distances, have each mic record a different combination of these 2 voices.
* Maybe speaker 1 is a little louder in mic 1 + speaker 2 is a little bit louder on mic 2, b/c the 2 mics are at different positions relative to the 2 speakers, but each mic would cause a separate overlapping combination of both voices.
* We can take these 2 mic recordings + give them to an unsupervised learning algorithm called the **cocktail party algorithm** + tell it to find structure in this data for us.
* The algorithm will “listen“ to these audio recordings + say “it sounds like 2 recordings are being added together/have been summed together to produce these recordings”
* Moreover, the cocktail party algorithm will, on its own*, separate out* these 2 audio sources that were being added/summed together to form other recordings
* It can then separate out 2 people speaking different languages into 2 separate recordings of each language, or separate out 1 person speaking + some background music into 2 separate recordings
* It might seem complicated to implement this + do a lot of coding of audio processing or link a bunch of synthesizer Java libraries that process audio
* But, it turns out the algorithm to do this can be done w/ 1 line of code, but it took researchers a LONG time to come up w/ this 1 line of code.
* **[W,s,v] = svd((repmat(sum(x.\*x,1),size(x,1),1).\*x)\*x’);**
* This is *not* an easy problem, but it turns out that when you use the right programming environment, many learning algorithms can be run w/ a really short program
* We will use the **Octave programming environment,** an open-source software (or MatLab)
* It turns out in Silicon Valley, for a lot of ML algorithms, what we do is 1st prototype software in Octave, b/c software in Octave makes it incredibly fast to implement learning algorithms.
* Above, the **SVD** function (**singular value decomposition**) is e a linear algebra routine built into Octave
* If trying to do this in C++ or Java, it would be many lines of code, linking complex C++ or Java libraries.
* Most ML algorithms learn much faster if using Octave as a programming environment and learning + prototyping tool, as it'll let one learn + prototype learning algorithms much more quickly.
* Many people in Silicon Valley companies use software like Octave to first prototype a ML algorithm, + only after getting it to work, then migrate it to C++ or Java, etc.
* By doing things this way, you can often get an algorithm to work much faster than if starting in C++
* **Unsupervised Learning =** a learning setting where you give an algorithm a ton of data + just ask it to find structure in the data for us.
* Ex: Use a clustering algorithm to cluster articles together about the same story
* Ex: Ask algorithm to discover market segments automatically based on a DB of customer data

Introduction Quiz

* A CPU program is said to learn from experience E with respect to some task T and some performance measure P if its performance on T, as measured by P, improves with experience E. Suppose we feed a learning algorithm a lot of historical weather data, and have it learn to predict weather. What would be a reasonable choice for P?
* **The probability of it correctly predicting a future date's weather.**
* **T = Predicting weather correctly**
* **E = Predicting the weather**
* Suppose you are working on weather prediction, and use a learning algorithm to predict tomorrow's temperature (in degrees C/F). Would you treat this as a classification or a regression problem?
* **Regression**
* Suppose you are working on stock market prediction. You’d like to predict whether or not a certain company will win a patent infringement lawsuit (by training on data of companies that had to defend against similar lawsuits). Would you treat this as a classification or a regression problem?
* **Classification**
* Some of the problems below are best addressed using a supervised learning algorithm, + others w/ an unsupervised learning algorithm. Which of the following would you apply supervised learning to? In each case, assume some appropriate dataset is available for your algorithm to learn from.
* **In farming, given data on crop yields over the last 50 years, predict next year's crop yields.**
* **Have a CPU examine an audio clip of a piece of music + classify whether or not there’re vocals (i.e., a human voice singing), or if it is a clip of only musical instruments (and no vocals)**
* Which of these is a reasonable definition of machine learning?
* **The field of study that gives CPU’s the ability to learn without being explicitly programmed.**
* An email program watches which emails we mark as spam or not spam, and based on this activity learns how to better filter spam. What is the task, T, here?
* **T = Classify emails as spam or not spam**
* **E = watching you classify spam and not spam**
* **P = probability the program will correctly ID /not spam, OR the # or fraction/% of emails correctly classified as spam/not spam**