Getting Set Up

Installation Checklist

- Install Enthought Canopy (version 1.6.2 or newer!)
- Open up the package manager, and install:
  - scikit_learn
  - xlrd
  - statsmodels
- Open an editor window and go to the interactive command prompt
  - !pip install pydot2

Python Basics

Let’s just jump right into some code.

Types of Data

Presented by
Frank Kane

Many Flavors of Data

Major Types of Data

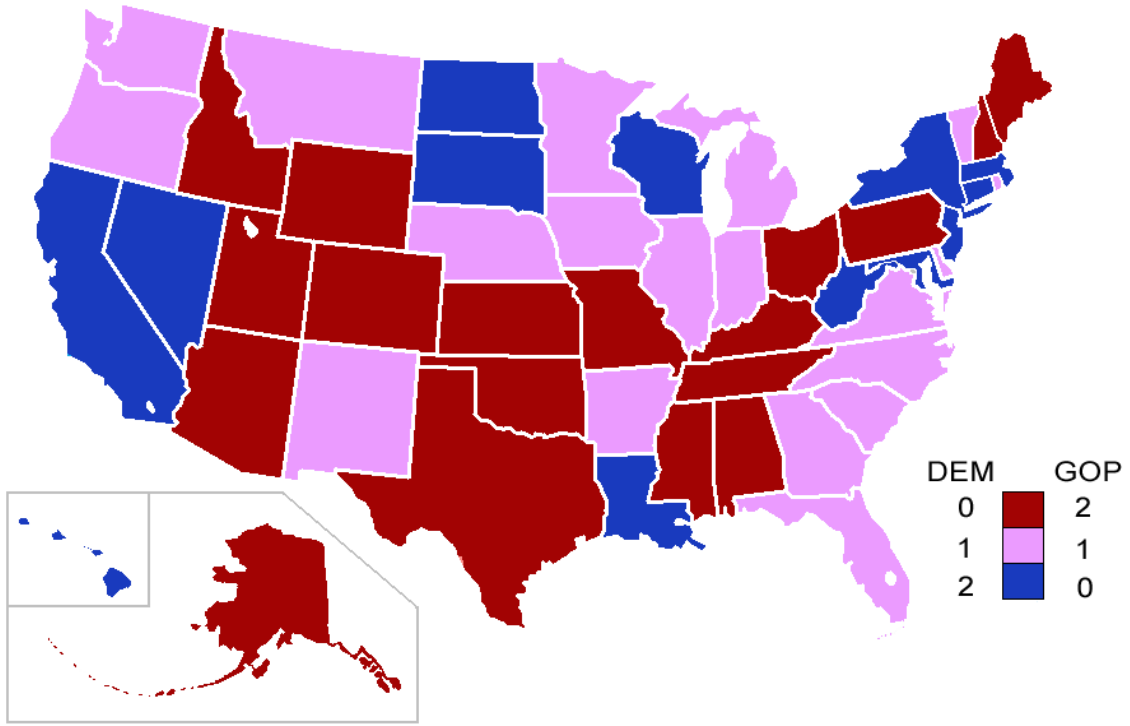
- Numerical
- Categorical
- Ordinal

Numerical

- Represents some sort of quantitative measurement
  - Heights of people, page load times, stock prices, etc.
- Discrete Data
  - Integer based; often counts of some event.
    - How many purchases did a customer make in a year?
    - How many times did I flip “heads”?
- Continuous Data
  - Has an infinite number of possible values
    - How much time did it take for a user to check out?
    - How much rain fell on a given day?

Categorical

- Qualitative data that has no inherent mathematical meaning
  - Gender, Yes/no (binary data), Race, State of Residence, Product Category, Political Party, etc.
- You can assign numbers to categories in order to represent them more compactly, but the numbers don’t have mathematical meaning

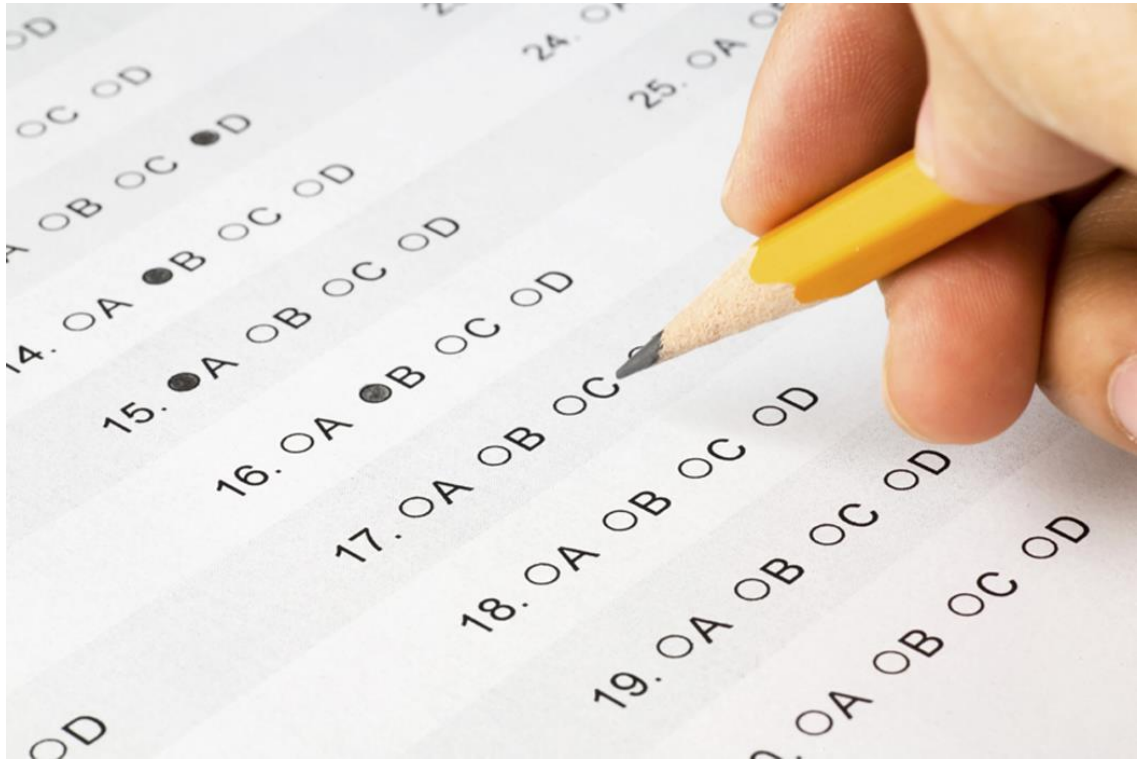


Ordinal

- A mixture of numerical and categorical
- Categorical data that has mathematical meaning
- Example: movie ratings on a 1-5 scale.
  - Ratings must be 1, 2, 3, 4, or 5
  - But these values have mathematical meaning; 1 means it's a worse movie than a 2.

Quiz time!

- Are the following types of data numerical, categorical, or ordinal?
  - How much gas is in your gas tank
  - A rating of your overall health where the choices are 1, 2, 3, or 4, corresponding to “poor”, “moderate”, “good”, and “excellent”
  - The races of your classmates
  - Ages in years
  - Money spent in a store



Mean, Median, and Mode

Mean

- AKA Average
- Sum / number of samples
- Example:
  - Number of children in each house on my street:

0, 2, 3, 2, 1, 0, 0, 2, 0

The MEAN is (0+2+3+2+1+0+0+2+0) / 9 = **1.11**

Median

- Sort the values, and take the value at the midpoint.
- Example:

0, 2, 3, 2, 1, 0, 0, 2, 0

Sort it:

0, 0, 0, 0, 1, 2, 2, 2, 3

Median

- If you have an even number of samples, take the average of the two in the middle.
- Median is less susceptible to outliers than the mean
  - Example: mean household income in the US is $72,641, but the median is only $51,939 – because the mean is skewed by a handful of billionaires.
  - Median better represents the “typical” American in this example.

Mode

- The most common value in a data set
  - Not relevant to continuous numerical data
- Back to our number of kids in each house example:

0, 2, 3, 2, 1, 0, 0, 2, 0

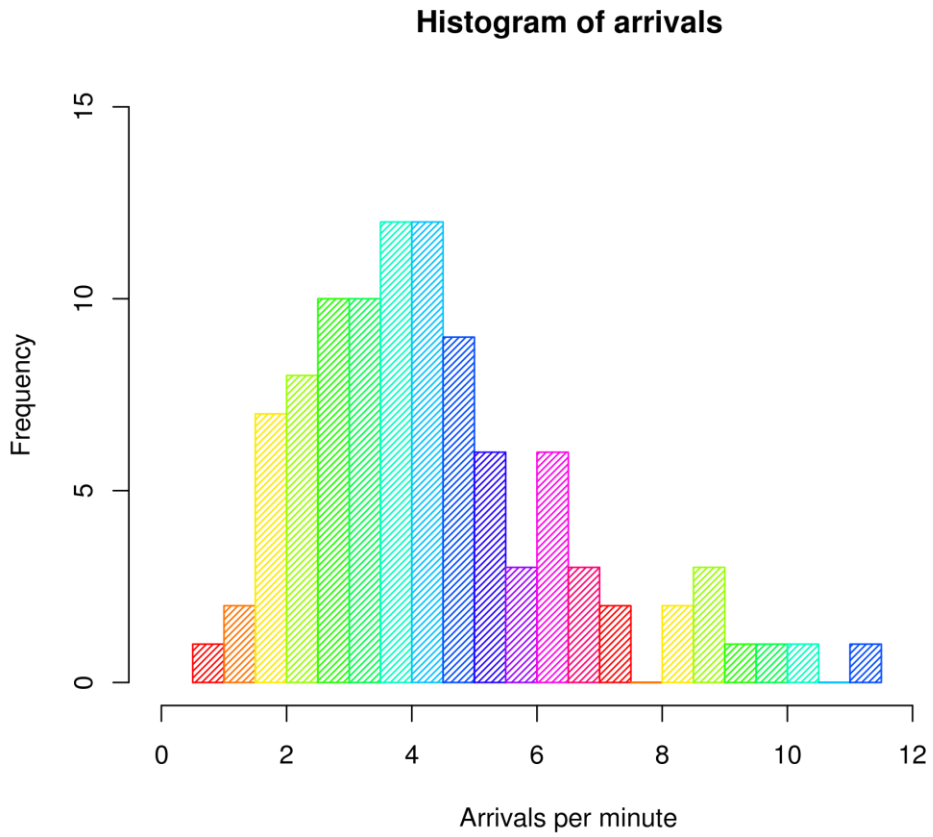
How many of each value are there?

0: 4, 1: 1, 2: 3, 3: 1

The MODE is 0

Standard Deviation and Variance

An example of a histogram



Variance measures how “spread-out” the data is.

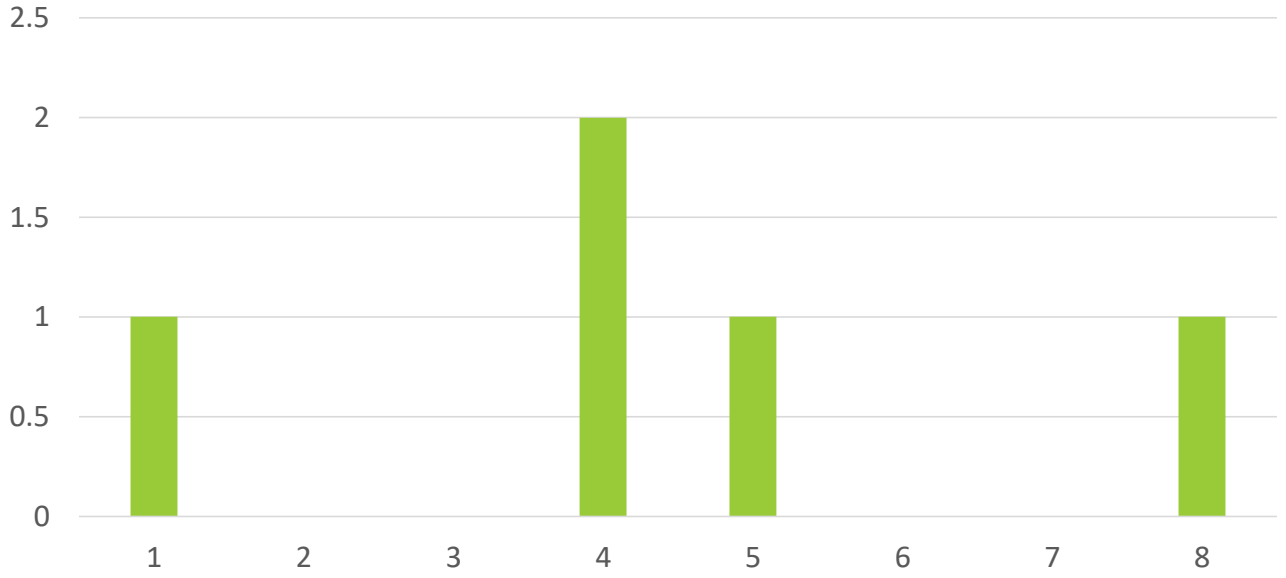
- Variance (σ^2) is simply the **average of the squared differences from the mean**
- Example: What is the variance of the data set (1, 4, 5, 4, 8)?
  - First find the mean: (1+4+5+4+8)/5 = 4.4
  - Now find the differences from the mean: (-3.4, -0.4, 0.6, -0.4, 3.6)
  - Find the squared differences: (11.56, 0.16, 0.36, 0.16, 12.96)
  - Find the average of the squared differences:
    - σ^2= (11.56 + 0.16 + 0.36 + 0.16 + 12.96) / 5 = 5.04

Standard Deviation σ is just the square root of the variance.

$\sigma^2 = 5.04$

$\sigma = \sqrt{5.04} = 2.24$

So the standard deviation of (1, 4, 5, 4, 8) is 2.24.



This is usually used as a way to identify outliers. Data points that lie more than one standard deviation from the mean can be considered unusual.

You can talk about how extreme a data point is by talking about "how many sigmas" away from the mean it is.

Population vs. Sample

- If you're working with a sample of data instead of an entire data set (the entire *population*)…
  - Then you want to use the "sample variance" instead of the "population variance"
  - For N samples, you just divide the squared variances by N-1 instead of N.
  - So, in our example, we computed the population variance like this:
    - σ^2= (11.56 + 0.16 + 0.36 + 0.16 + 12.96) / 5 = 5.04
  - But the sample variance would be:
    - S^2= (11.56 + 0.16 + 0.36 + 0.16 + 12.96) / 4 = 6.3

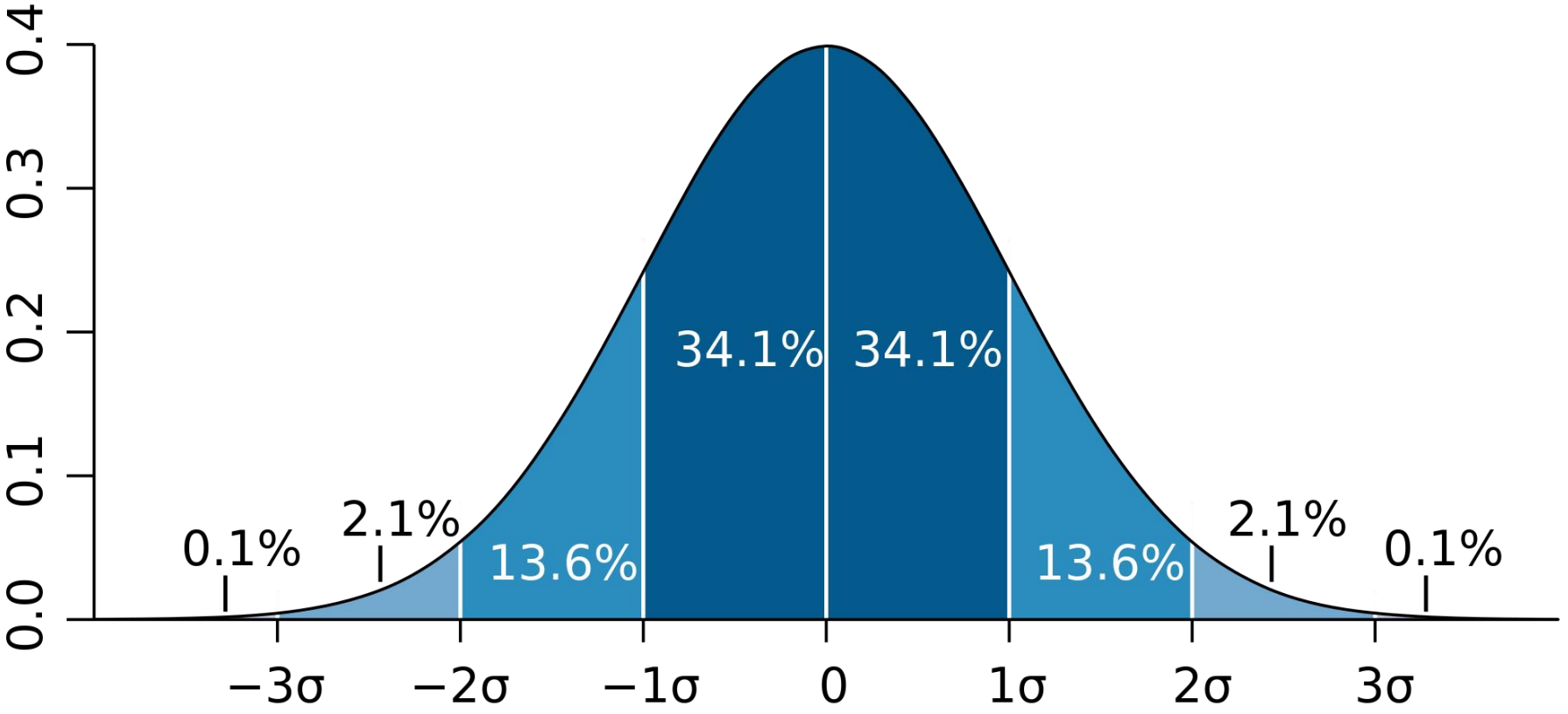
Fancy Math

- Population variance:
  - $\sigma^2 = \frac{\sum(X-\mu)^2}{N}$
- Sample variance:
  - $s^2 = \frac{\sum(X-M)^2}{N-1}$

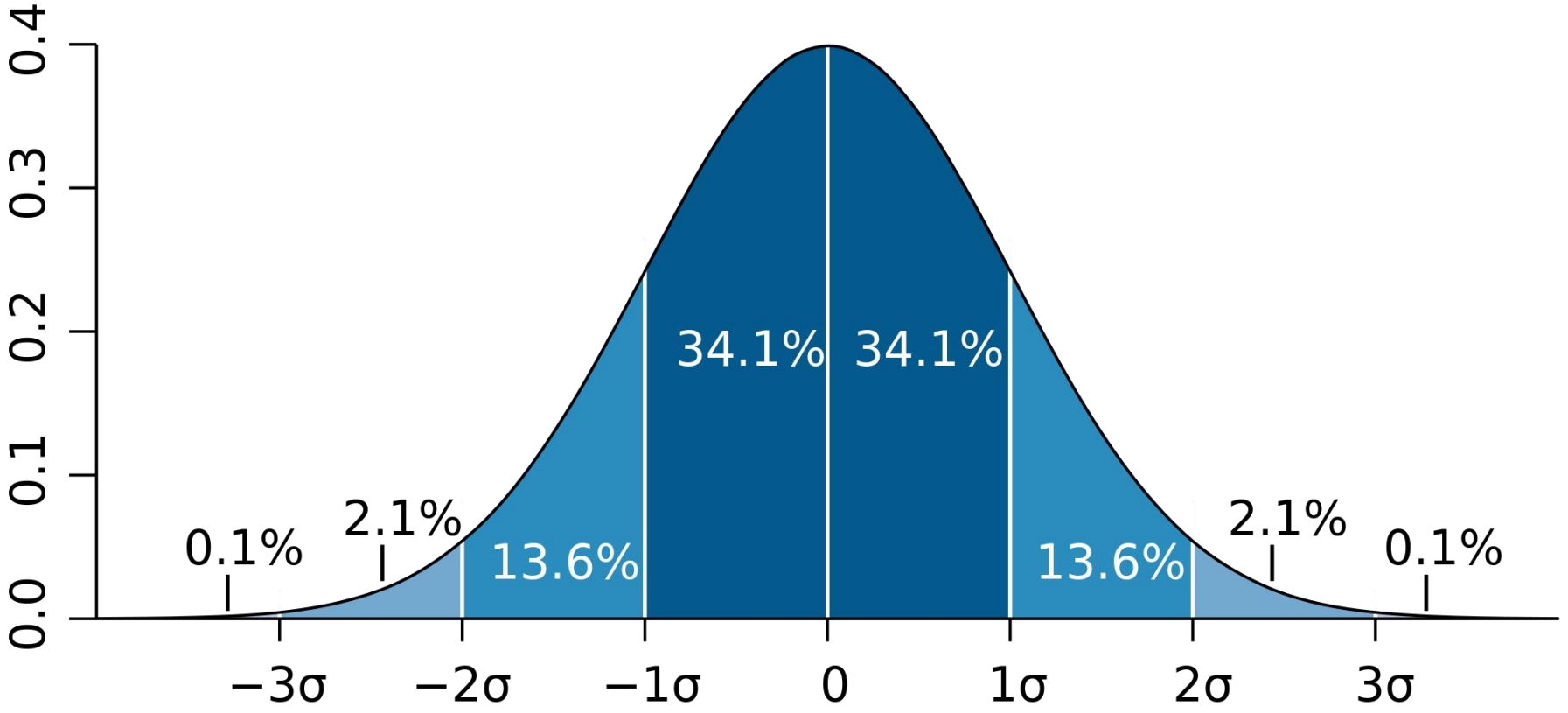
Let’s look at another example.

Probability Density Functions

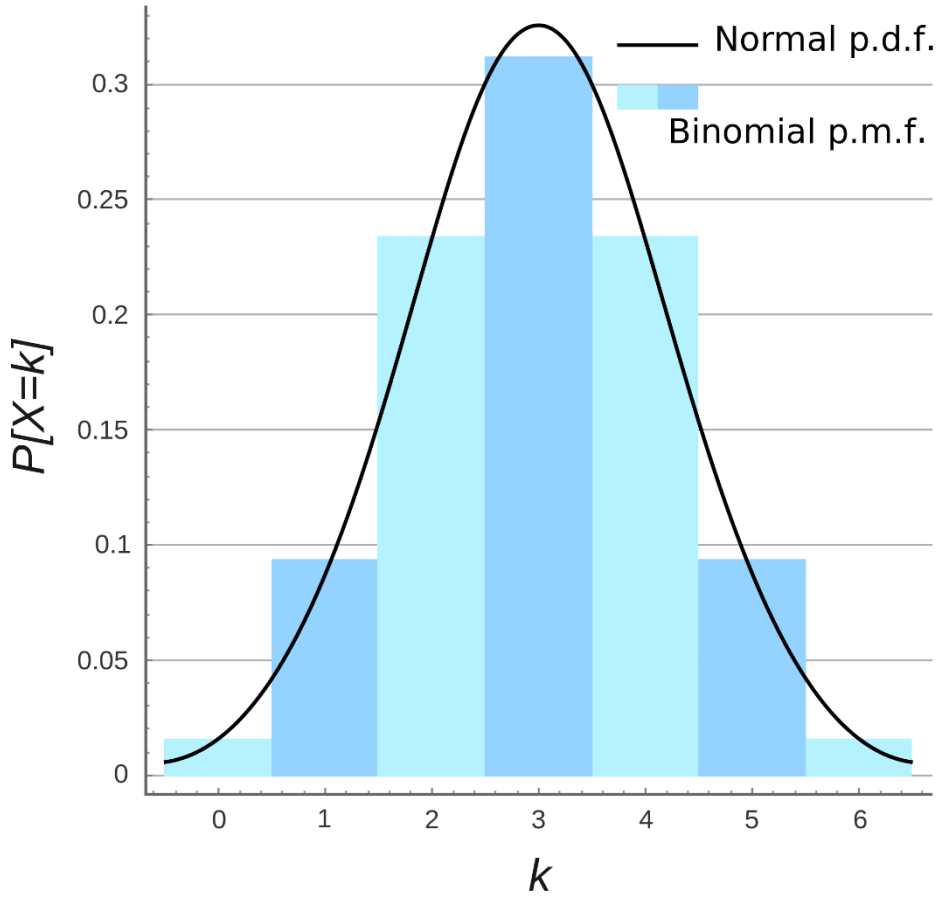
Example: a “normal distribution”



Gives you the probability of a data point falling within some given range of a given value.



Probability Mass Function

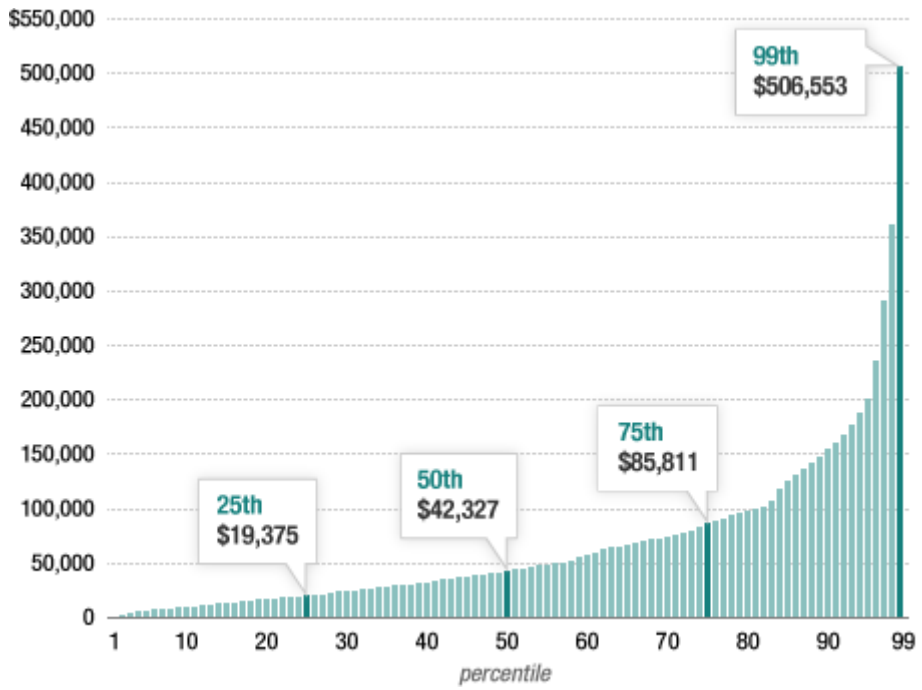


Let’s play with some examples.

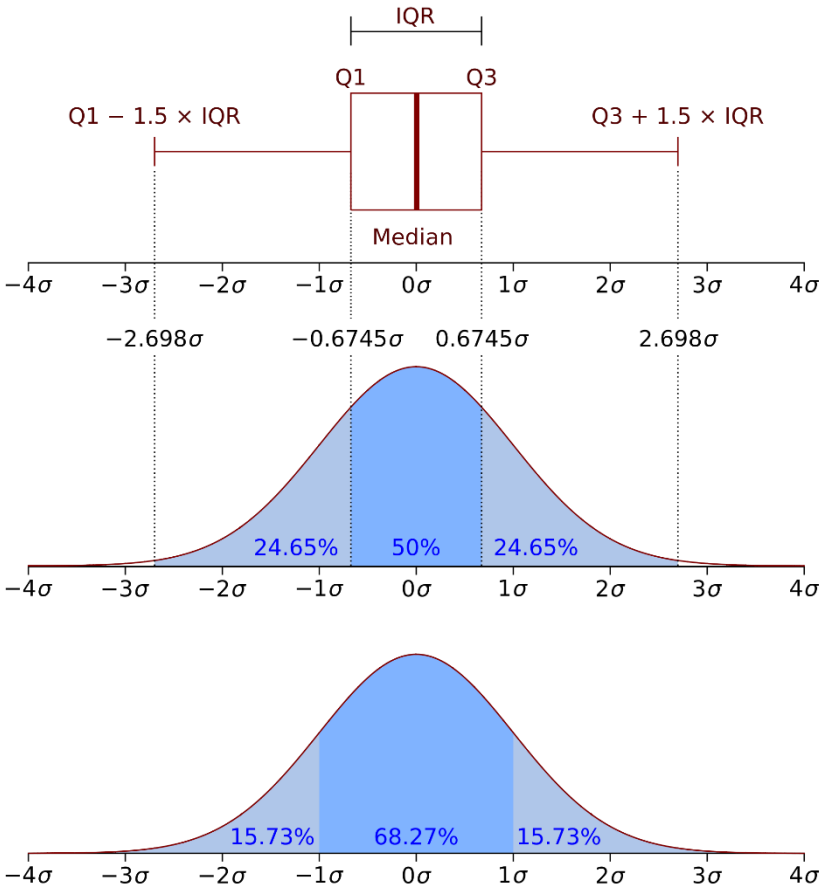
Percentiles and Moments

Percentiles

- In a data set, what's the point at which X% of the values are less than that value?
- Example: income distribution



Percentiles in a normal distribution



Let’s look at some examples.

Moments

- Quantitative measures of the shape of a probability density function
- Mathematically they are a bit hard to wrap your head around:
  - $\mu_n = \int_{-\infty}^{\infty} (x - c)^n f(x) dx$ (for moment n around value c)
- But intuitively, it's a lot simpler in statistics.

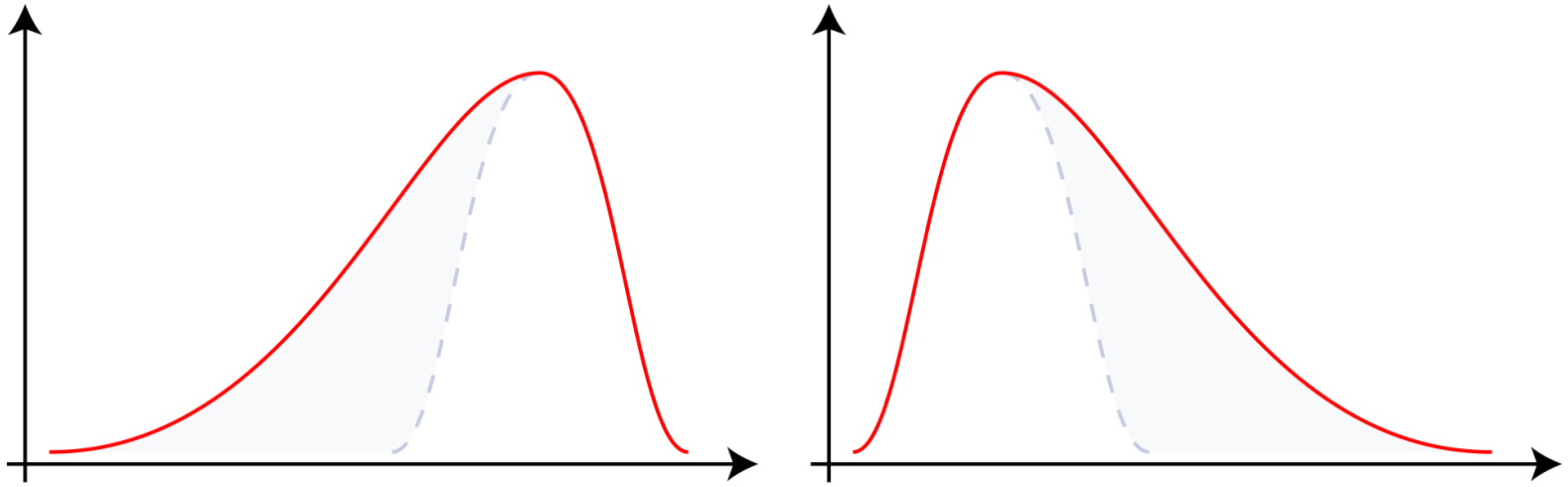
The first moment is the mean.

The second moment is the variance.

Yes, it’s just that simple.

The third moment is “skew” (γ)

- How “lopsided” is the distribution?
- A distribution with a longer tail on the left will be skewed left, and have a negative skew.

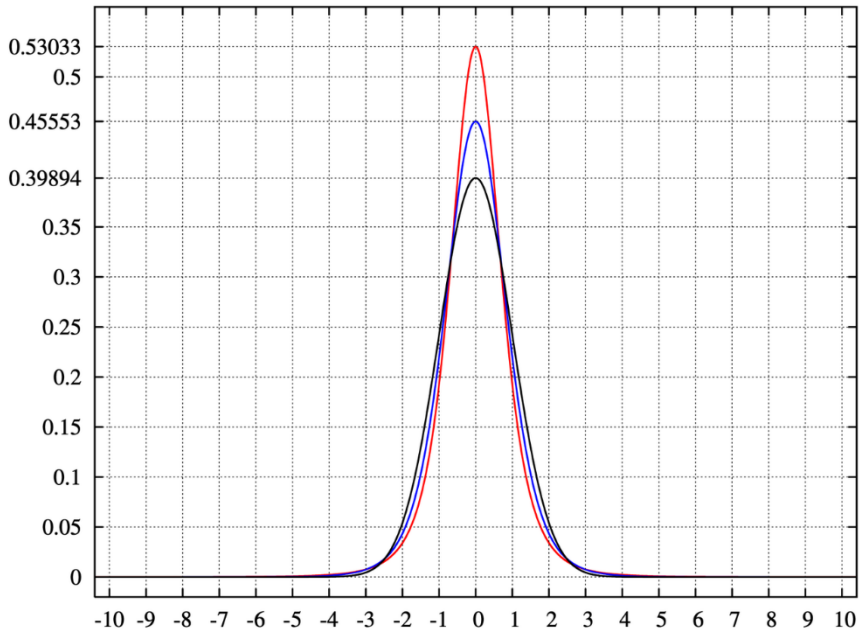


Negative Skew

Positive Skew

The fourth moment is “kurtosis”

- How thick is the tail, and how sharp is the peak, compared to a normal distribution?
- Example: higher peaks have higher kurtosis

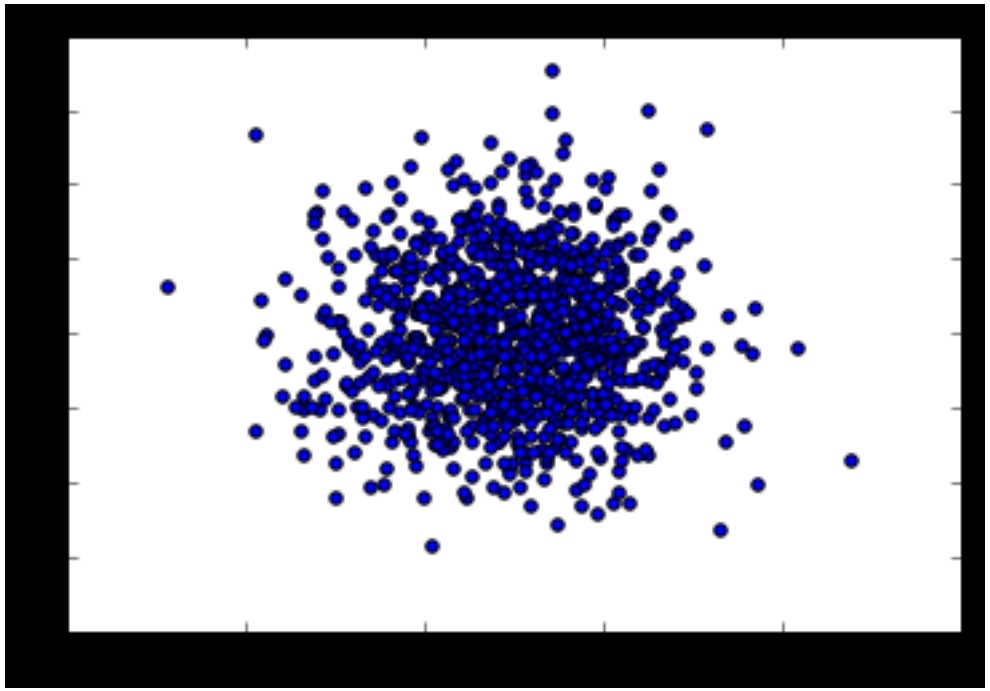


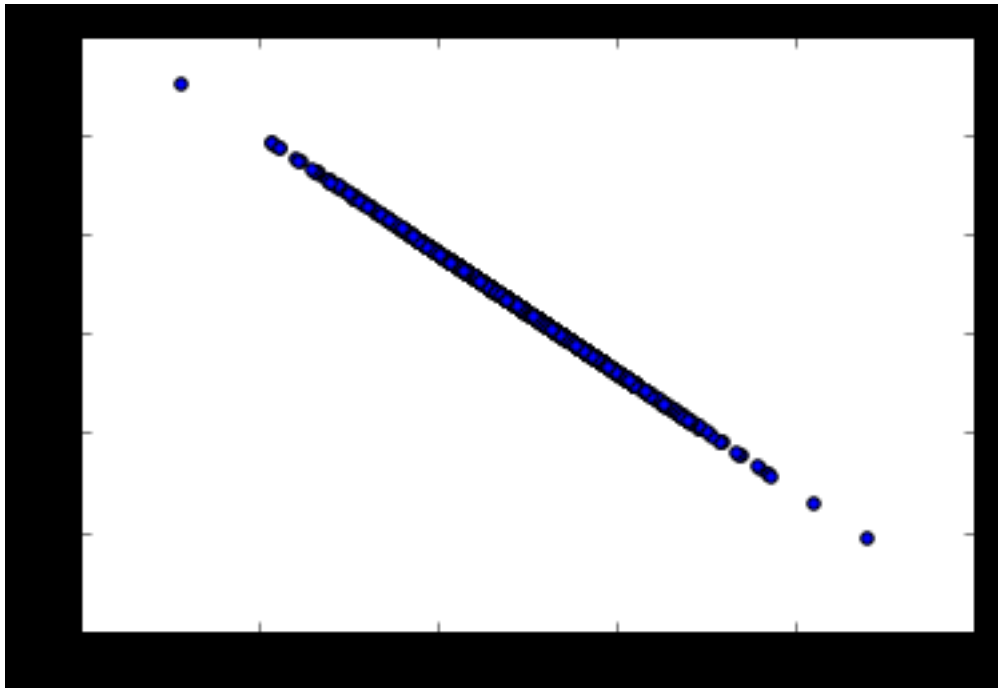
Let’s compute the 4 moments in Python.

Covariance and Correlation

Covariance

- Measures how two variables vary in tandem from their means.





Measuring covariance

- Think of the data sets for the two variables as high-dimensional vectors
- Convert these to vectors of variances from the mean
- Take the dot product (cosine of the angle between them) of the two vectors
- Divide by the sample size

Interpreting covariance is hard

- We know a small covariance, close to 0, means there isn't much correlation between the two variables.
- And large covariances – that is, far from 0 (could be negative for inverse relationships) mean there is a correlation
- But how large is "large"?

That’s where correlation comes in!

- Just divide the covariance by the standard deviations of both variables, and that normalizes things.
- So a correlation of -1 means a perfect inverse correlation
- Correlation of 0: no correlation
- Correlation 1: perfect correlation

Remember: correlation does not imply causation!

- Only a controlled, randomized experiment can give you insights on causation.
- Use correlation to decide what experiments to conduct!

Let’s play with some data.

Conditional Probability

Conditional Probability

- If I have two events that depend on each other, what's the probability that both will occur?
- Notation: P(A,B) is the probability of A and B both occurring
- P(B|A) : Probability of B given that A has occurred
- We know:

$$P(B|A) = \frac{P(A,B)}{P(A)}$$

For example

- I give my students two tests. 60% of my students passed both tests, but the first test was easier – 80% passed that one. What percentage of students who passed the first test also passed the second?
- A = passing the first test, B = passing the second test
- So we are asking for P(B|A) – the probability of B given A
- $P(B|A) = \frac{P(A,B)}{P(A)} = \frac{0.6}{0.8} = 0.75$
- 75% of students who passed the first test passed the second.

Let’s do another example using Python.

Bayes' Theorem

Bayes' Theorem

- Now that you understand conditional probability, you can understand Bayes' Theorem:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

In English – the probability of A given B, is the probability of A times the probability of B given A over the probability of B.

The key insight is that the probability of something that depends on B depends very much on the base probability of B and A. People ignore this all the time.

Bayes' Theorem to the rescue

- Drug testing is a common example. Even a "highly accurate" drug test can produce more false positives than true positives.
- Let's say we have a drug test that can accurately identify users of a drug 99% of the time, and accurately has a negative result for 99% of non-users. But only 0.3% of the overall population actually uses this drug.

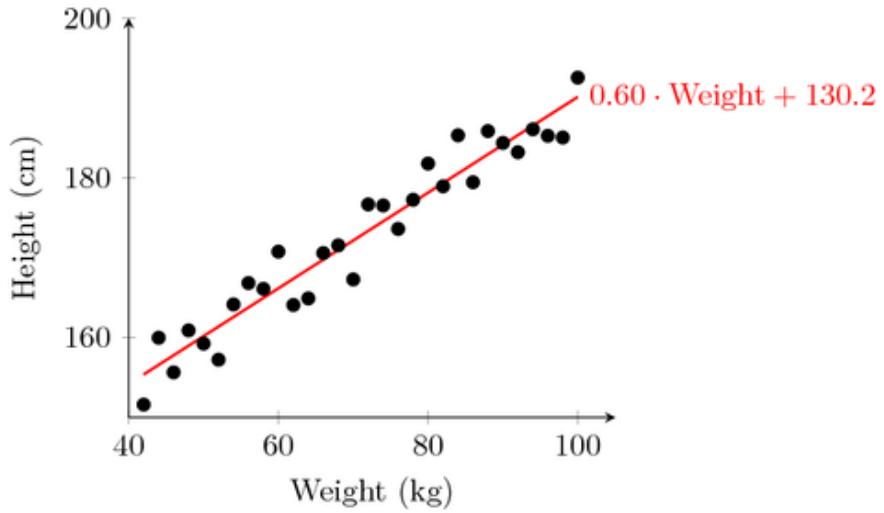
Bayes' Theorem to the rescue

- Event A = Is a user of the drug, Event B = tested positively for the drug.
- We can work out from that information that P(B) is 1.3% (0.99 * 0.003 + 0.01 * 0.997 – the probability of testing positive if you do use, plus the probability of testing positive if you don't.)
- $P(A|B) = \frac{P(A)P(B|A)}{P(B)} = \frac{0.003 * 0.99}{0.013} = 22.8\%$
- So the odds of someone being an actual user of the drug given that they tested positive is only 22.8%!
- Even though P(B|A) is high (99%), it doesn't mean P(A|B) is high.

Linear Regression

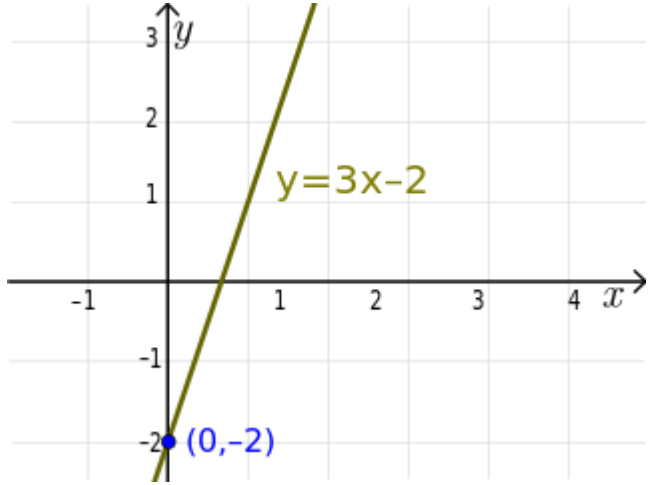
Linear Regression

- Fit a line to a data set of observations
- Use this line to predict unobserved values
- I don’t know why they call it “regression.” It’s really misleading. You can use it to predict points in the future, the past, whatever. In fact time usually has nothing to do with it.



Linear Regression: How does it work?

- Usually using “least squares”
- Minimizes the squared-error between each point and the line
- Remember the slope-intercept equation of a line? y=mx+b
- The slope is the correlation between the two variables times the standard deviation in Y, all divided by the standard deviation in X.
  - Neat how standard deviation how some real mathematical meaning, eh?
- The intercept is the mean of Y minus the slope times the mean of X
- But Python will do all that for you.

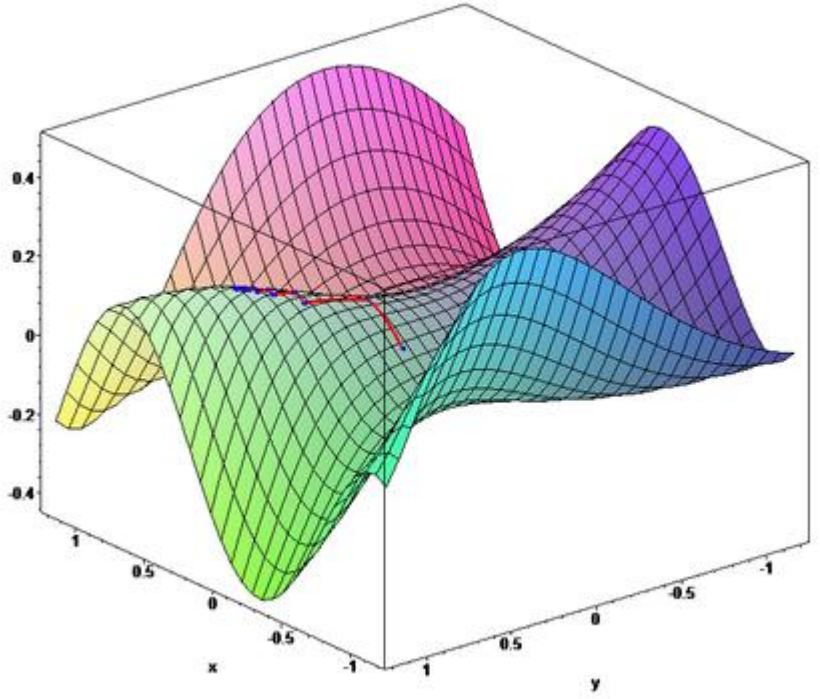


Linear Regression: How does it work?

- Least squares minimizes the sum of squared errors.
- This is the same as maximizing the likelihood of the observed data if you start thinking of the problem in terms of probabilities and probability distribution functions
- This is sometimes called “maximum likelihood estimation”

More than one way to do it

- Gradient Descent is an alternate method to least squares.
- Basically iterates to find the line that best follows the contours defined by the data.
- Can make sense when dealing with 3D data
- Easy to try in Python and just compare the results to least squares
  - But usually least squares is a perfectly good choice.



Measuring error with r-squared

- How do we measure how well our line fits our data?
- R-squared (aka coefficient of determination) measures:

The fraction of the total variation in Y that is captured by the model

Computing r-squared

$$1.0 - \frac{sum\ of\ squared\ errors}{sum\ of\ squared\ variation\ from\ mean}$$

Interpreting r-squared

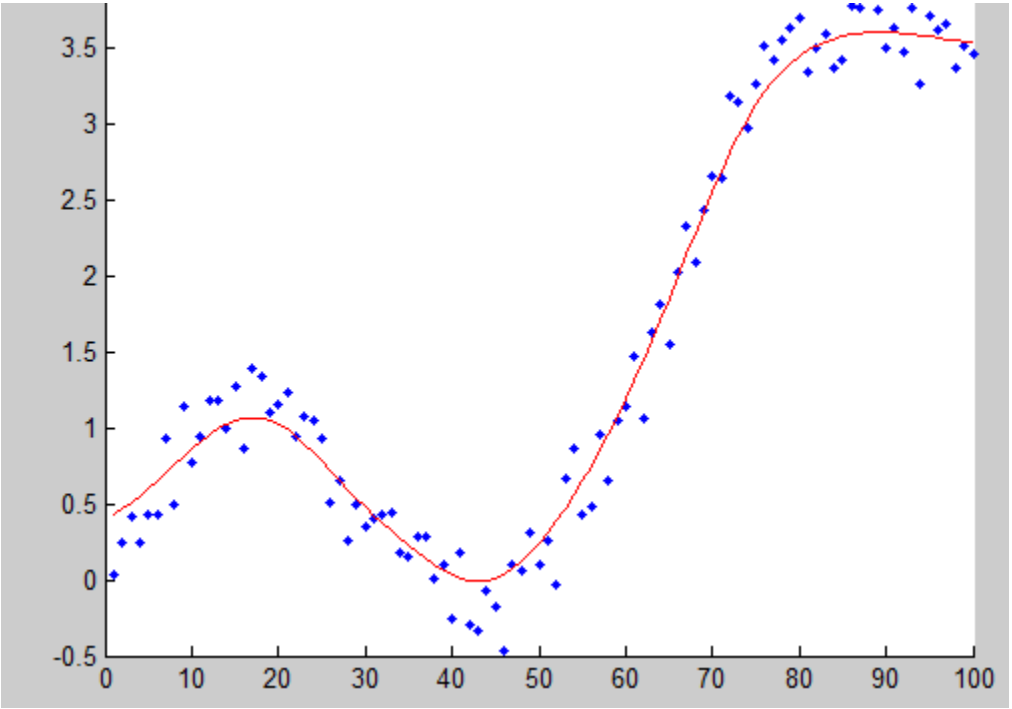
- Ranges from 0 to 1
- 0 is bad (none of the variance is captured), 1 is good (all of the variance is captured).

Let’s play with an example.

Polynomial Regression

Why limit ourselves to straight lines?

- Not all relationships are linear.
- Linear formula: y = mx + b
  - This is a “first order” or “first degree” polynomial, as the power of x is 1
- Second order polynomial: $y = ax^2 + bx + c$
- Third order: $y = ax^3 + bx^2 + cx + d$
- Higher orders produce more complex curves.



Beware overfitting

- Don’t use more degrees than you need
- Visualize your data first to see how complex of a curve there might really be
- Visualize the fit – is your curve going out of its way to accommodate outliers?
- A high r-squared simply means your curve fits your *training data* well; but it may not be a good predictor.
- Later we’ll talk about more principled ways to detect overfitting (train/test)

Let’s play with an example

- numpy.polyfit() makes it easy.

Multivariate Regression

Multivariate regression (Multiple Regression)

- What if more than one variable influences the one you're interested in?
- Example: predicting a price for a car based on its many attributes (body style, brand, mileage, etc.)

Still uses least squares

- We just end up with coefficients for each factor.
  - For example, $price = \alpha + \beta_1 \text{mileage} + \beta_2 \text{age} + \beta_3 \text{doors}$
  - These coefficients imply how important each factor is (if the data is all normalized!)
  - Get rid of ones that don't matter!
- Can still measure fit with r-squared
- Need to assume the different factors are not themselves dependent on each other.

Let’s dive into an example.

- The statsmodel package makes it easy.

Multi-Level Models

Multi-Level Models

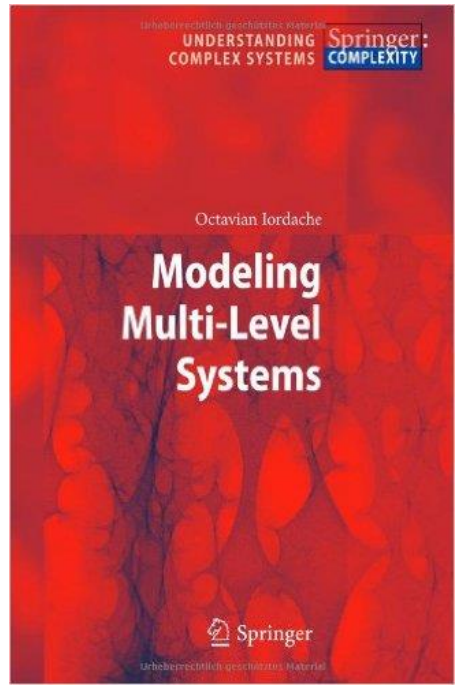
- The concept is that some effects happen at various levels.
- Example: your health depends on a hierarchy of the health of your cells, organs, you as a whole, your family, your city, and the world you live in.
- Your wealth depends on your own work, what your parents did, what your grandparents did, etc.
- Multi-level models attempt to model and account for these interdependencies.

Modeling multiple levels

- You must identify the factors that affect the outcome you're trying to predict at each level.
- For example – SAT scores might be predicted based on the genetics of individual children, the home environment of individual children, the crime rate of the neighborhood they live in, the quality of the teachers in their school, the funding of their school district, and the education policies of their state.
- Some of these factors affect more than one level. For example, crime rate might influence the home environment too.

Doing this is hard.

- I just want you to be aware of the concept, as multi-level models showed up on some data science job requirements I've seen.
- You're not ready for it yet. Entire advanced statistics and modeling courses exist on this one topic alone.
- Thick books exist on it too, when you're ready.



Supervised and Unsupervised Machine Learning

And the concept of train/test

What is machine learning?

- Algorithms that can learn from observational data, and can make predictions based on it.

Yeah, that’s pretty much what your own brain does too.

Unsupervised Learning

- The model is not given any “answers” to learn from; it must make sense of the data just given the observations themselves.
- Example: group (cluster) some objects together into 2 different sets. But I don’t tell you what the “right” set is for any object ahead of time.




Do I want big and small things? Round and square things? Red and blue things? Unsupervised learning could give me any of those results.

Unsupervised Learning

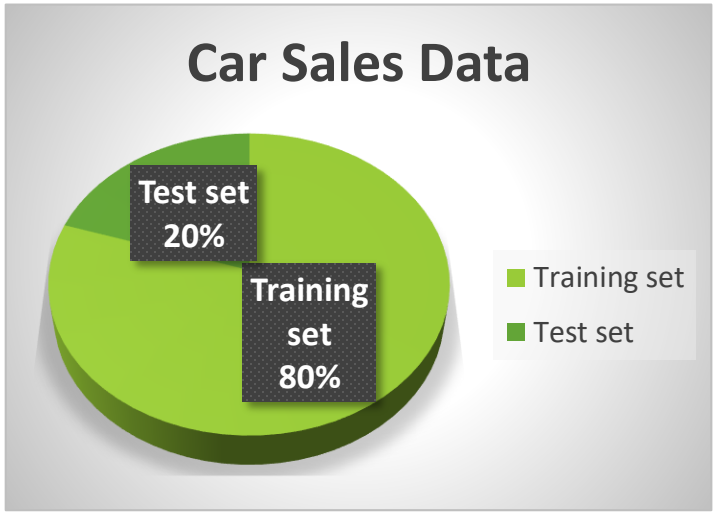
- Unsupervised learning sounds awful! Why use it?
- Maybe you don't know what you're looking for – you're looking for *latent variables*.
- Example: clustering users on a dating site based on their information and behavior. Perhaps you'll find there are groups of people that emerge that don't conform to your known stereotypes.
- Cluster movies based on their properties. Perhaps our current concepts of genre are outdated?
- Analyze the text of product descriptions to find the terms that carry the most meaning for a certain category.

Supervised Learning

- In supervised learning, the data the algorithm "learns" from comes with the "correct" answers.
- The model created is then used to predict the answer for new, unknown values.
- Example: You can *train* a model for predicting car prices based on car attributes using historical sales data. That model can then *predict* the optimal price for new cars that haven't been sold before.

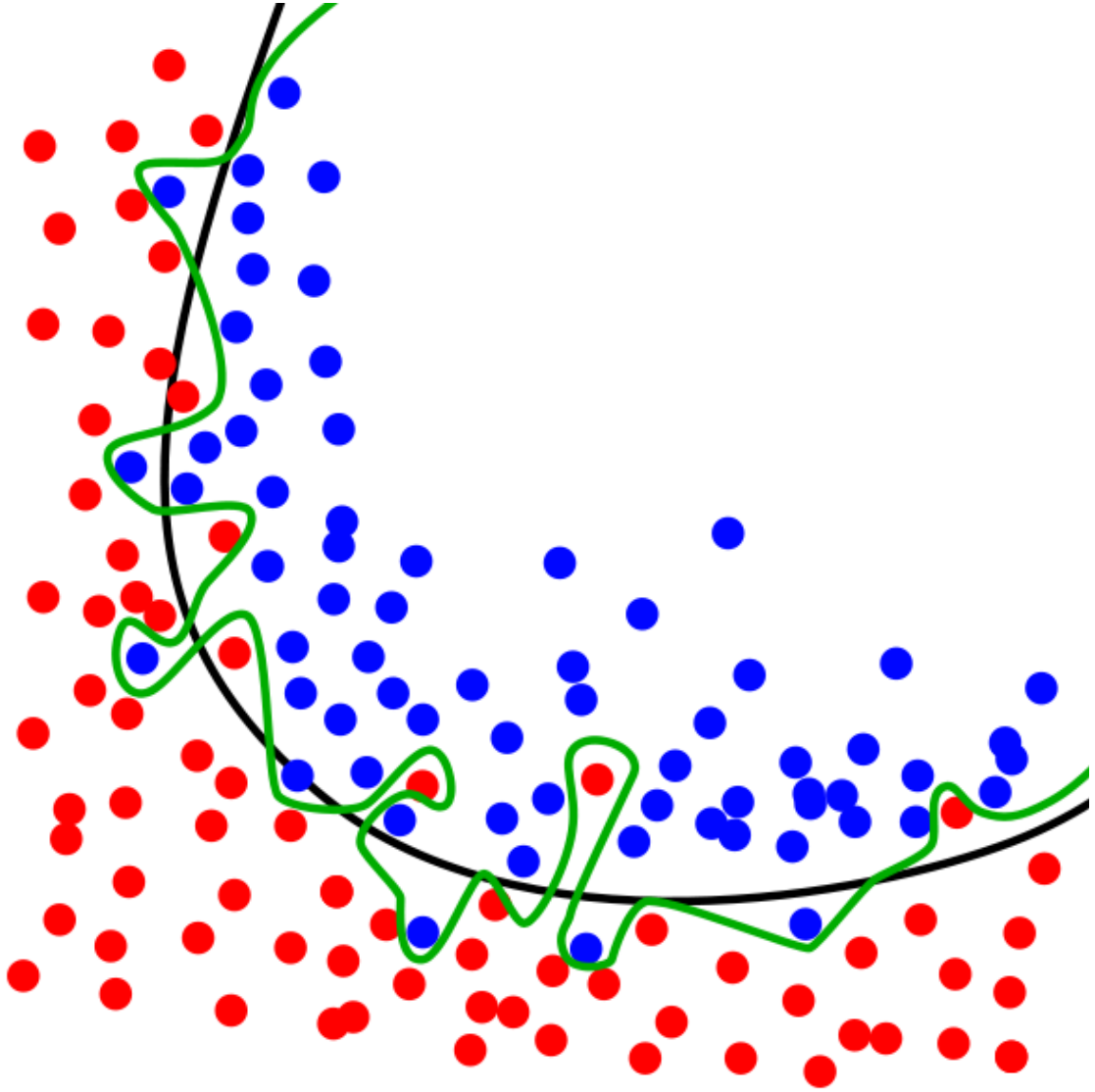
Evaluating Supervised Learning

- If you have a set of training data that includes the value you're trying to predict – you don't have to guess if the resulting model is good or not.
- If you have enough training data, you can split it into two parts: a *training* set and a *test* set.
- You then train the model using only the training set
- And then measure (using r-squared or some other metric) the model's accuracy by asking it to predict values for the test set, and compare that to the known, true values.



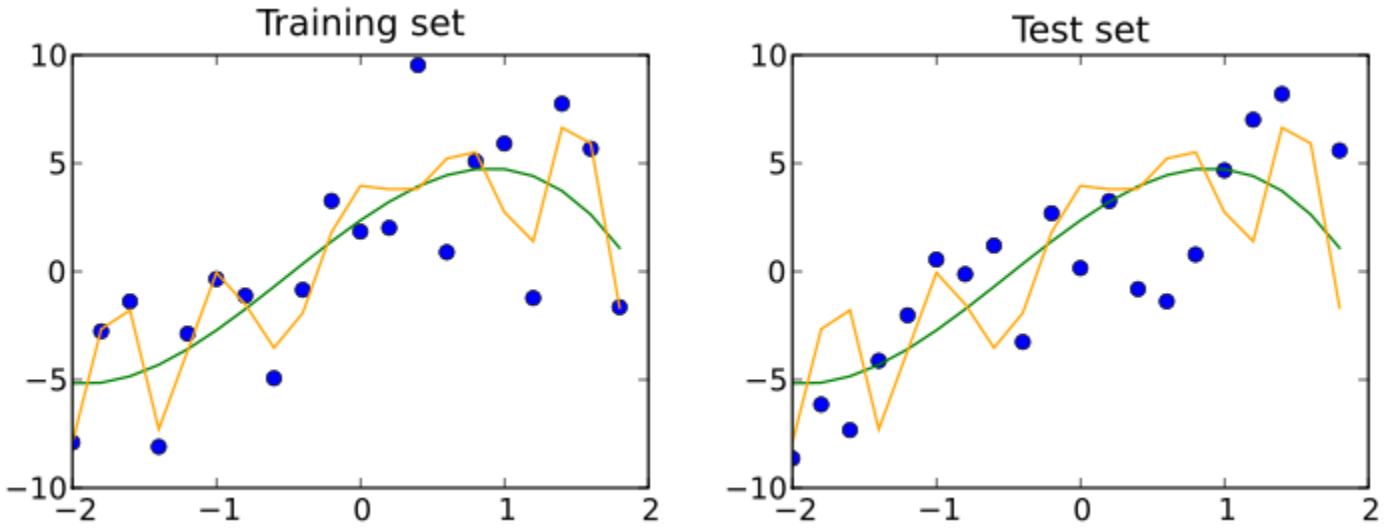
Train / Test in practice

- Need to ensure both sets are large enough to contain representatives of all the variations and outliers in the data you care about
- The data sets must be selected randomly
- Train/test is a great way to guard against *overfitting*



Train/Test is not Infallible

- Maybe your sample sizes are too small
- Or due to random chance your train and test sets look remarkably similar
- Overfitting can still happen



K-fold Cross Validation

- One way to further protect against overfitting is *K-fold cross validation*
- Sounds complicated. But it's a simple idea:
  - Split your data into K randomly-assigned segments
  - Reserve one segment as your test data
  - Train on each of the remaining K-1 segments and measure their performance against the test set
  - Take the average of the K-1 r-squared scores

Let’s go do some training and testing.

Bayesian Methods

Remember Bayes' Theorem?

- $P(A|B) = \frac{P(A)P(B|A)}{P(B)}$
- Let's use it for machine learning! I want a spam classifier.
- Example: how would we express the probability of an email being spam if it contains the word "free"?
- $P(Spam \mid Free) = \frac{P(Spam)P(Free \mid Spam)}{P(Free)}$
- The numerator is the probability of a message being spam and containing the word "free" (this is subtly different from what we're looking for)
- The denominator is the overall probability of an email containing the word "free". (Equivalent to P(Free|Spam)P(Spam) + P(Free|Not Spam)P(Not Spam))
- So together – this ratio is the % of emails with the word "free" that are spam.

What about all the other words?

- We can construct P(Spam | Word) for every (meaningful) word we encounter during training
- Then multiply these together when analyzing a new email to get the probability of it being spam.
- Assumes the presence of different words are independent of each other – one reason this is called “Naïve Bayes”.

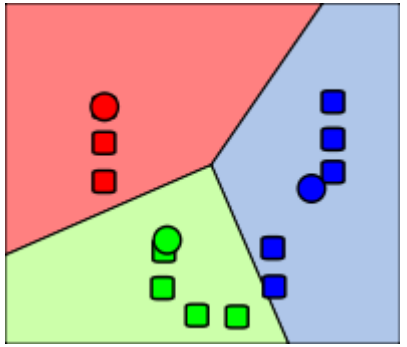
Sounds like a lot of work.

- Scikit-learn to the rescue!
- The CountVectorizer lets us operate on lots of words at once, and MultinomialNB does all the heavy lifting on Naïve Bayes.
- We'll train it on known sets of spam and "ham" (non-spam) emails
  - So this is supervised learning!
- Let's do this

K-Means Clustering

K-Means Clustering

- Attempts to split data into K groups that are closest to K centroids
- Unsupervised learning – uses only the positions of each data point
- Can uncover interesting groupings of people / things / behavior
  - Example: Where do millionaires live?
  - What genres of music / movies / etc. naturally fall out of data?
  - Create your own stereotypes from demographic data



K-Means Clustering

- Sounds fancy! Wow! Unsupervised machine learning! Clusters! K!
- Actually how it works is really simple.
  - Randomly pick K centroids (k-means)
  - Assign each data point to the centroid it’s closest to
  - Recompute the centroids based on the average position of each centroid’s points
  - Iterate until points stop changing assignment to centroids
- If you want to predict the cluster for new points, just find the centroid they’re closest to.

Graphical example

Images from Wikimedia Commons

K-Means Clustering Gotchas

- Choosing K
  - Try increasing K values until you stop getting large reductions in squared error (distances from each point to their centroids)
- Avoiding local minima
  - The random choice of initial centroids can yield different results
  - Run it a few times just to make sure your initial results aren’t wacky
- Labeling the clusters
  - K-Means does not attempt to assign any meaning to the clusters you find
  - It’s up to you to dig into the data and try to determine that

Let’s cluster stuff.

- Again, scikit-learn makes this easy.

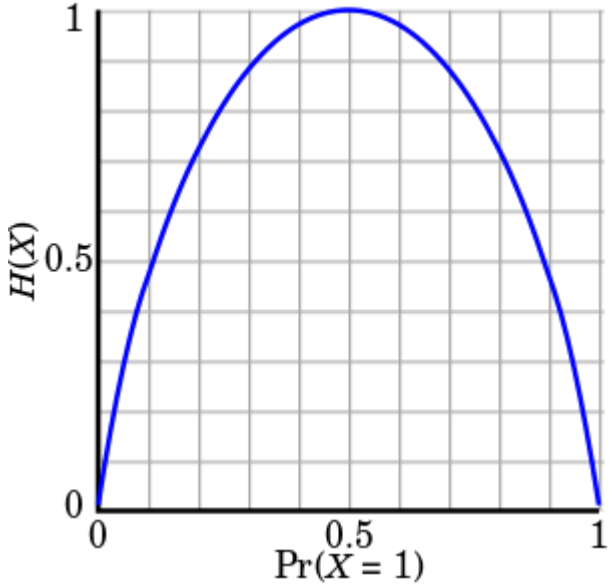
Entropy

Entropy

- A measure of a data set's disorder – how same or different it is.
- If we classify a data set into N different classes (example: a data set of animal attributes and their species)
  - The entropy is 0 if all of the classes in the data are the same (everyone is an iguana)
  - The entropy is high if they're all different
- Again, a fancy word for a simple concept.

Computing entropy

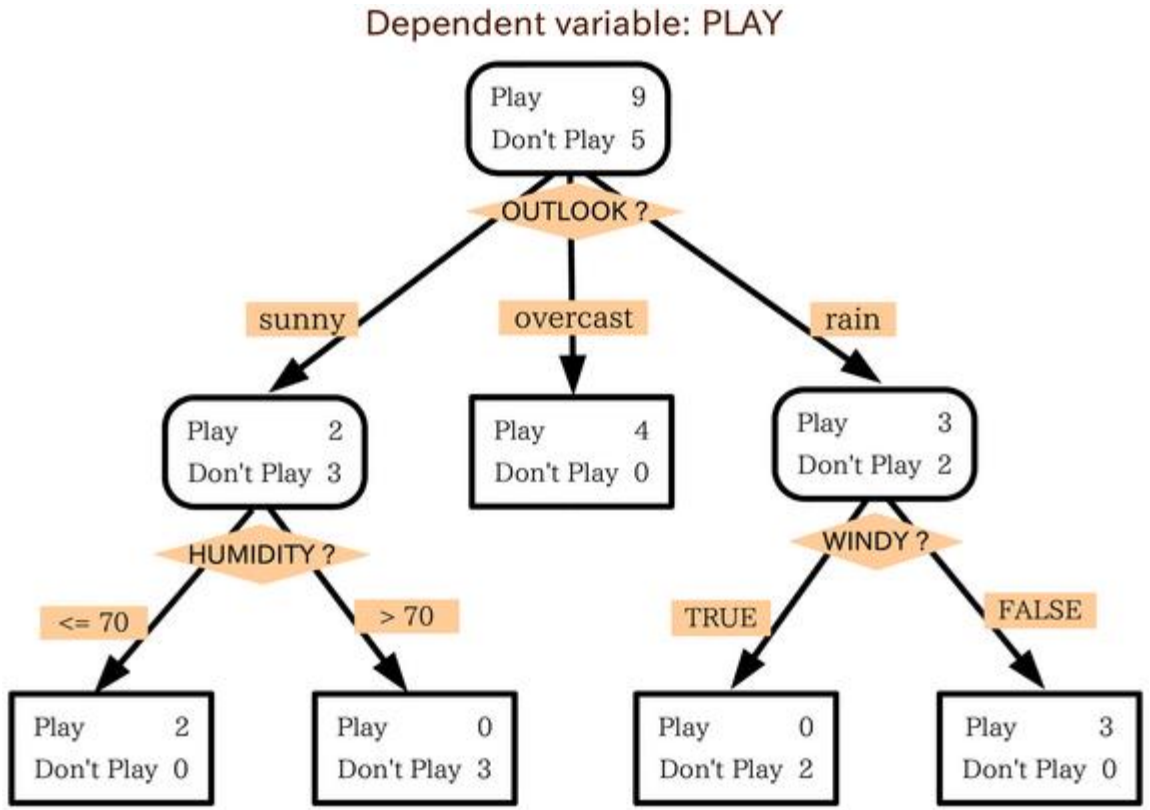
- $H(S) = -p_1 \ln p_1 - \cdots - p_n \ln p_n$
- p_i represents the proportion of the data labeled for each class
- Each term looks like this:



Decision Trees

Decision Trees

- You can actually construct a flowchart to help you decide a classification for something with machine learning
- This is called a Decision Tree
- Another form of supervised learning
  - Give it some sample data and the resulting classifications
  - Out comes a tree!



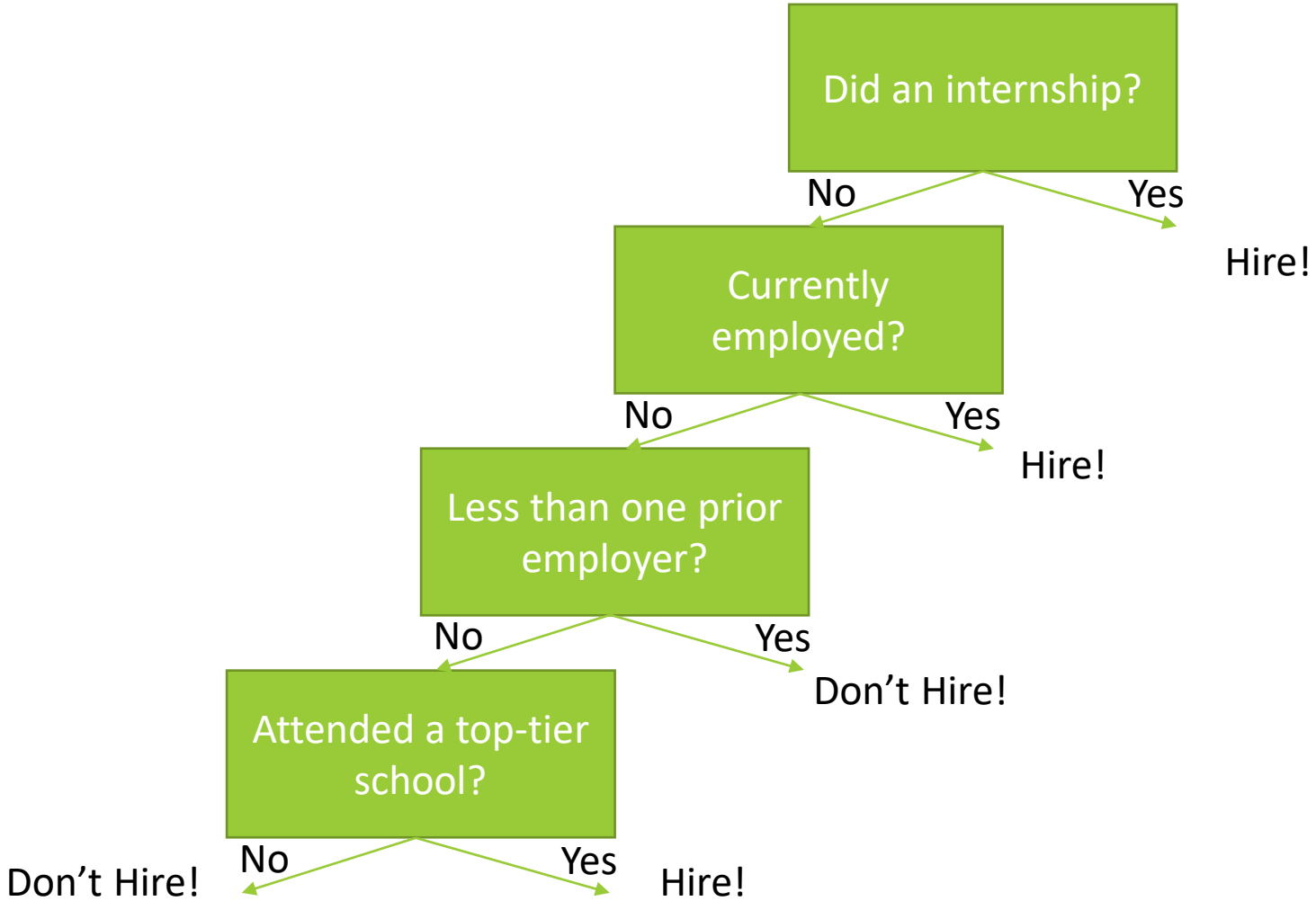
Decision Tree example

- You want to build a system to filter out resumes based on historical hiring data
- You have a database of some important attributes of job candidates, and you know which ones were hired and which ones weren’t
- You can train a decision tree on this data, and arrive at a system for predicting whether a candidate will get hired based on it!

Totally Fabricated Hiring Data

Candidate ID	Years Experience	Employed?	Previous employers	Level of Education	Top-tier school	Interned	Hired
0	10	1	4	0	0	0	1
1	0	0	0	0	1	1	1
2	7	0	6	0	0	0	0
3	2	1	1	1	1	0	1
4	20	0	2	2	1	0	0

Totally Fabricated Should-I-Hire-This-Person Tree



How Decision Trees Work

- At each step, find the attribute we can use to partition the data set to minimize the *entropy* of the data at the next step
- Fancy term for this simple algorithm: ID3
- It is a *greedy algorithm* – as it goes down the tree, it just picks the decision that reduce entropy the most at that stage.
  - That might not actually result in an optimal tree.
  - But it works.

Random Forests

- Decision trees are very susceptible to overfitting
- To fight this, we can construct several alternate decision trees and let them “vote” on the final classification
  - Randomly re-sample the input data for each tree (fancy term for this: *bootstrap aggregating* or *bagging*)
  - Randomize a subset of the attributes each step is allowed to choose from

Let’s go make some trees.

- Yet again, scikit-learn is awesome for this.

Ensemble Learning

Ensemble Learning

- Random Forests was an example of *ensemble learning*
- It just means we use multiple models to try and solve the same problem, and let them vote on the results.

Ensemble Learning

- Random Forests uses *bagging* (bootstrap aggregating) to implement ensemble learning
  - Many models are built by training on randomly-drawn subsets of the data
- *Boosting* is an alternate technique where each subsequent model in the ensemble boosts attributes that address data mis-classified by the previous model
- A *bucket of models* trains several different models using training data, and picks the one that works best with the test data
- *Stacking* runs multiple models at once on the data, and combines the results together
  - This is how the Netflix prize was won!

Advanced Ensemble Learning: Ways to Sound Smart

- Bayes Optimal Classifier
  - Theoretically the best – but almost always impractical
- Bayesian Parameter Averaging
  - Attempts to make BOC practical – but it's still misunderstood, susceptible to overfitting, and often outperformed by the simpler bagging approach
- Bayesian Model Combination
  - Tries to address all of those problems
  - But in the end, it's about the same as using cross-validation to find the best combination of models

Support Vector Machines

Support Vector Machines

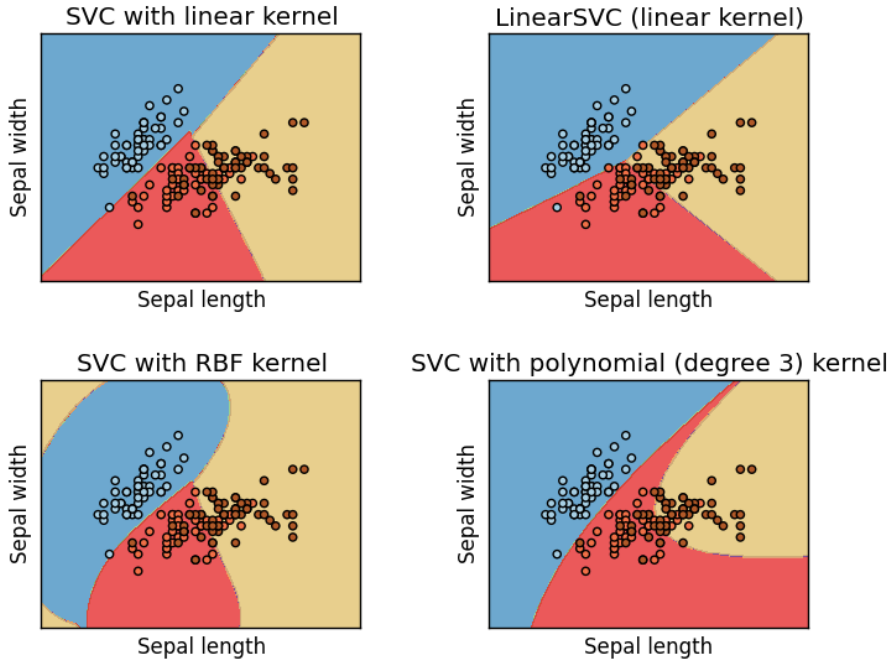
- Works well for classifying higher-dimensional data (lots of features)
- Finds higher-dimensional *support vectors* across which to divide the data (mathematically, these support vectors define hyperplanes. Needless to say I'm not going to get into the mathematical details!)
- Uses something called the *kernel trick* to represent data in higher-dimensional spaces to find hyperplanes that might not be apparent in lower dimensions

Higher dimensions? Hyperplanes? Huh?

- The important point is that SVM's employ some advanced mathematical trickery to cluster data, and it can handle data sets with lots of features.
- It's also fairly expensive – the "kernel trick" is the only thing that makes it possible.

Support Vector Classification

- In practice you'll use something called SVC to classify data using SVM.
- You can use different "kernels" with SVC. Some will work better than others for a given data set.

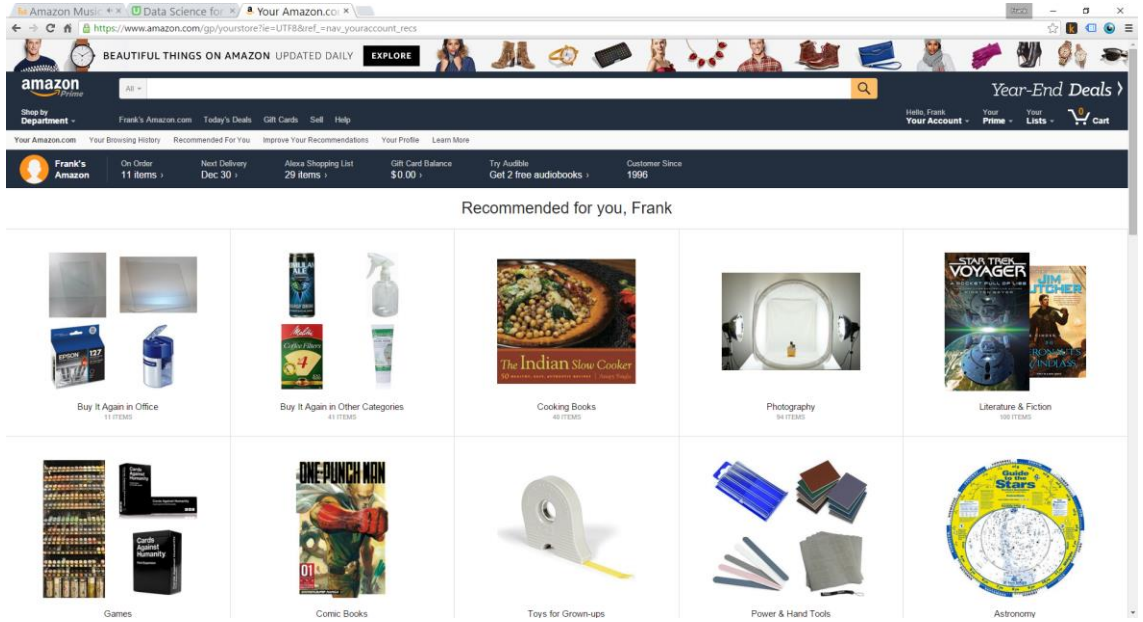


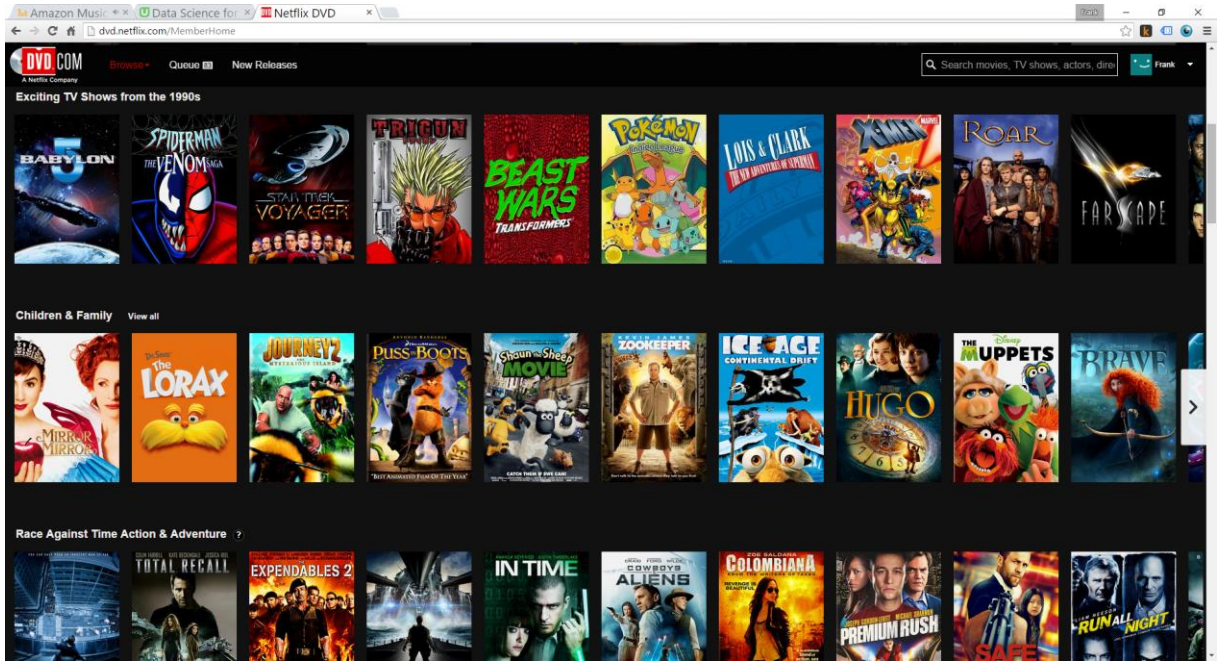
Let’s play with SVC’s

- Don’t even try to do this without scikit-learn.

Recommender Systems

What are recommender systems?

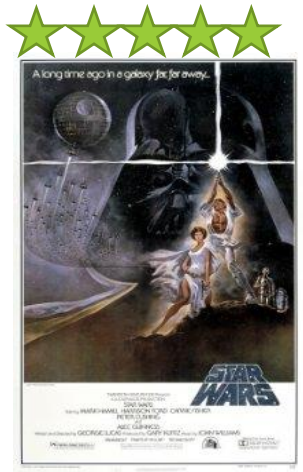


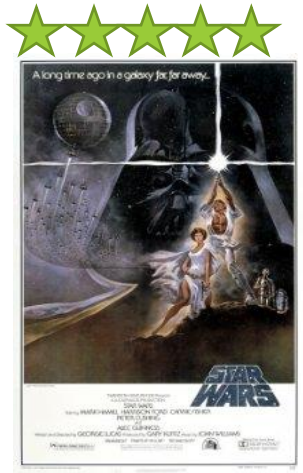


User-Based Collaborative Filtering

- Build a matrix of things each user bought/viewed/rated
- Compute similarity scores between users
- Find users similar to you
- Recommend stuff they bought/viewed/rated that you haven’t yet.

User-Based Collaborative Filtering





User-Based Collaborative Filtering

Problems with User-Based CF

- People are fickle; tastes change
- There are usually many more people than things
- People do bad things

What if we based recommendations on relationships between things instead of people?

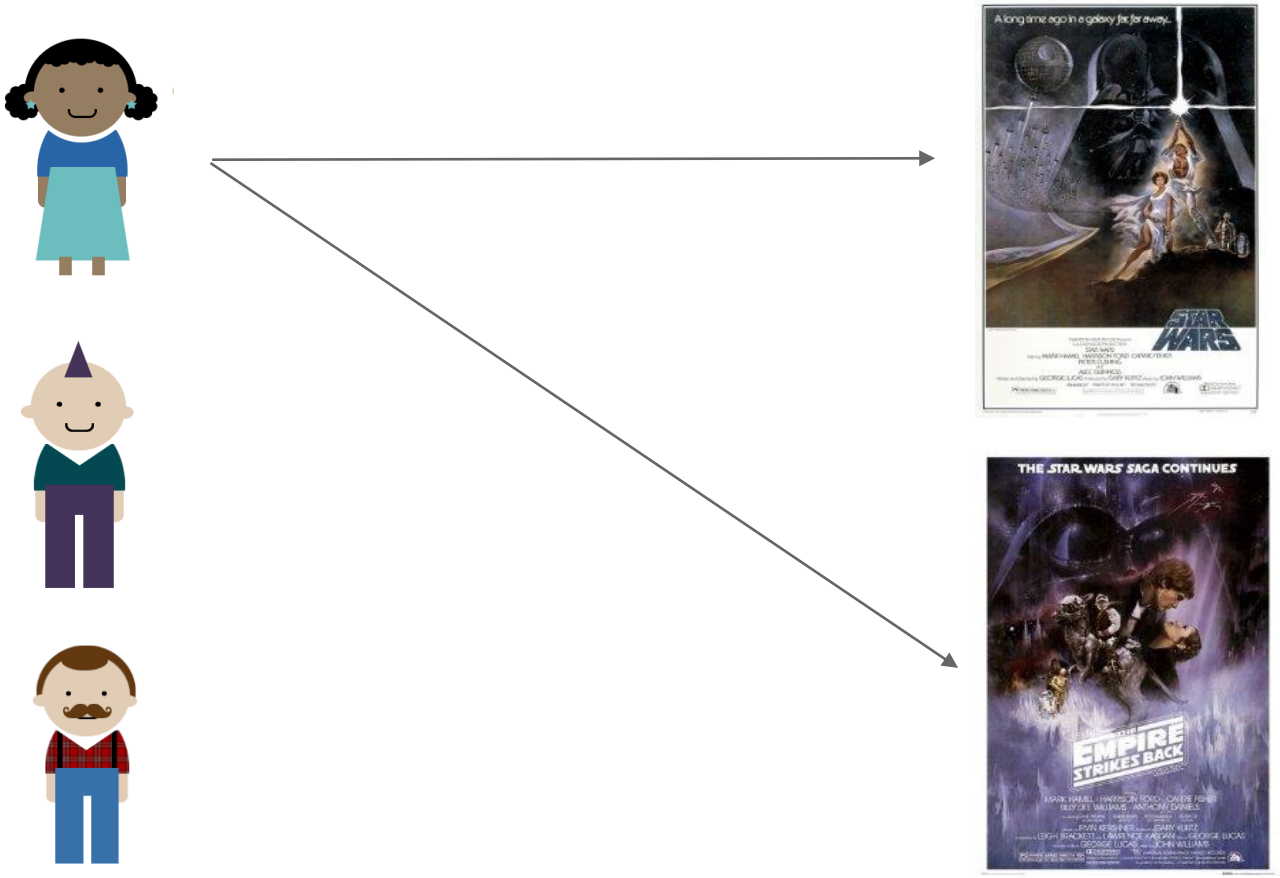
- A movie will always be the same movie – it doesn't change
- There are usually fewer things than people (less computation to do)
- Harder to game the system

Item-Based Collaborative Filtering

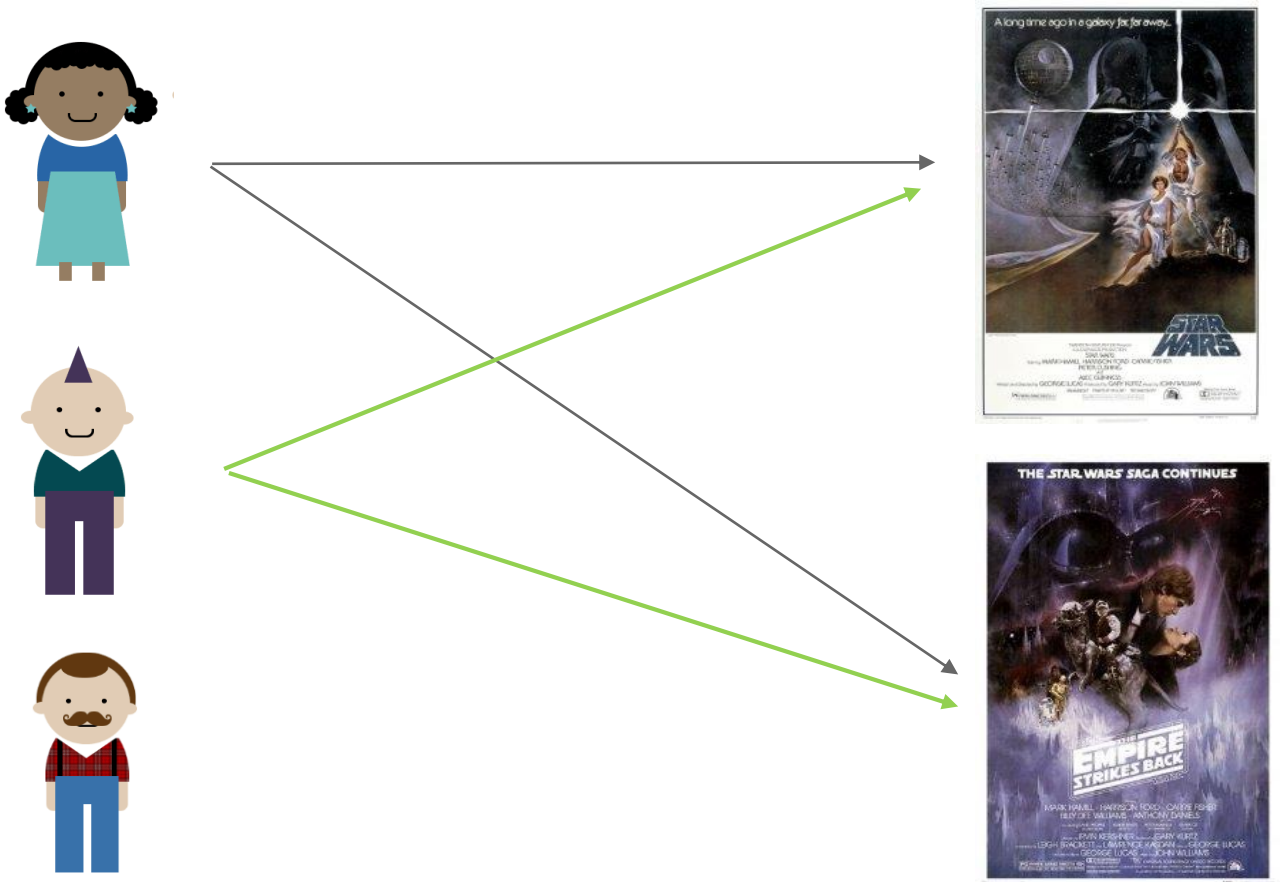
- Find every pair of movies that were watched by the same person
- Measure the similarity of their ratings across all users who watched both
- Sort by movie, then by similarity strength

- (This is just one way to do it!)

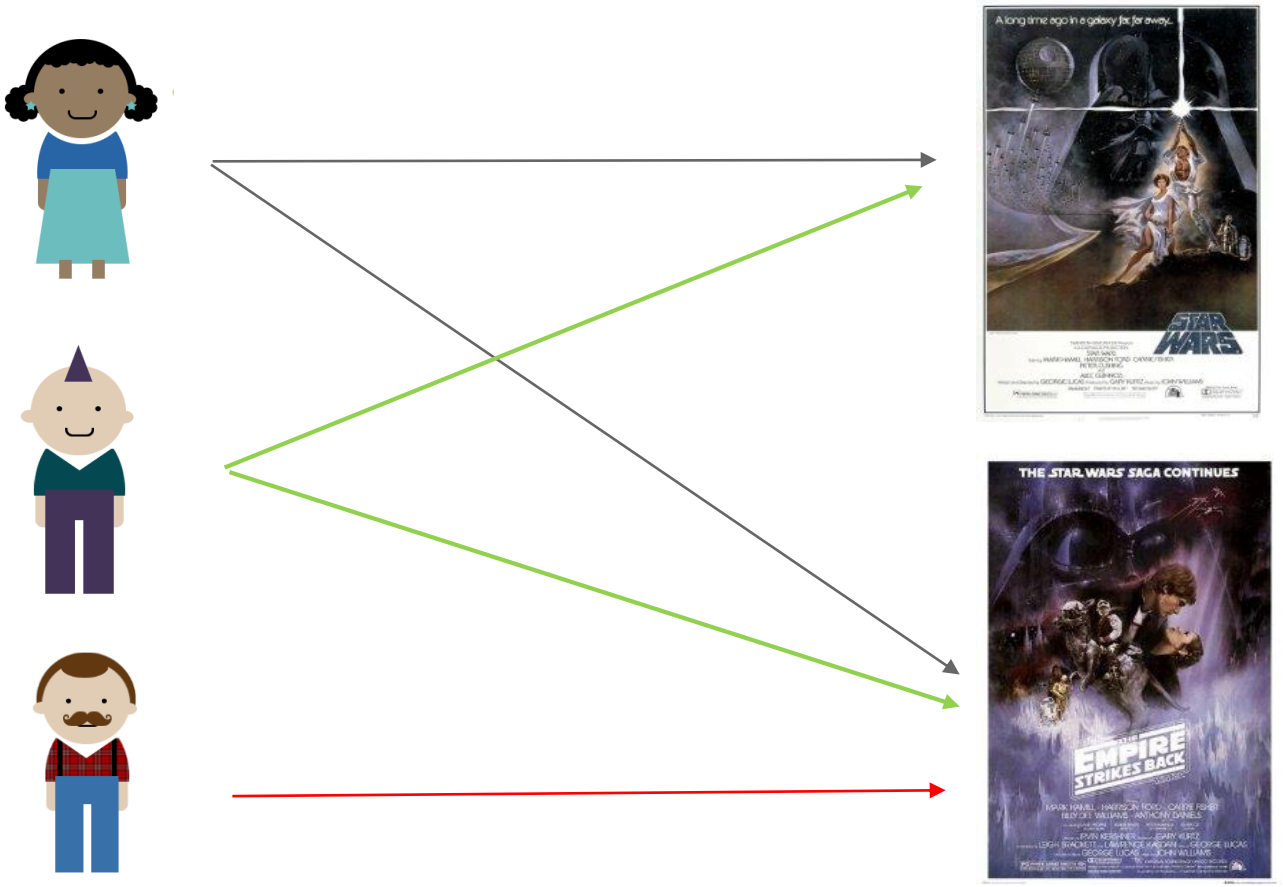
Item-Based Collaborative Filtering



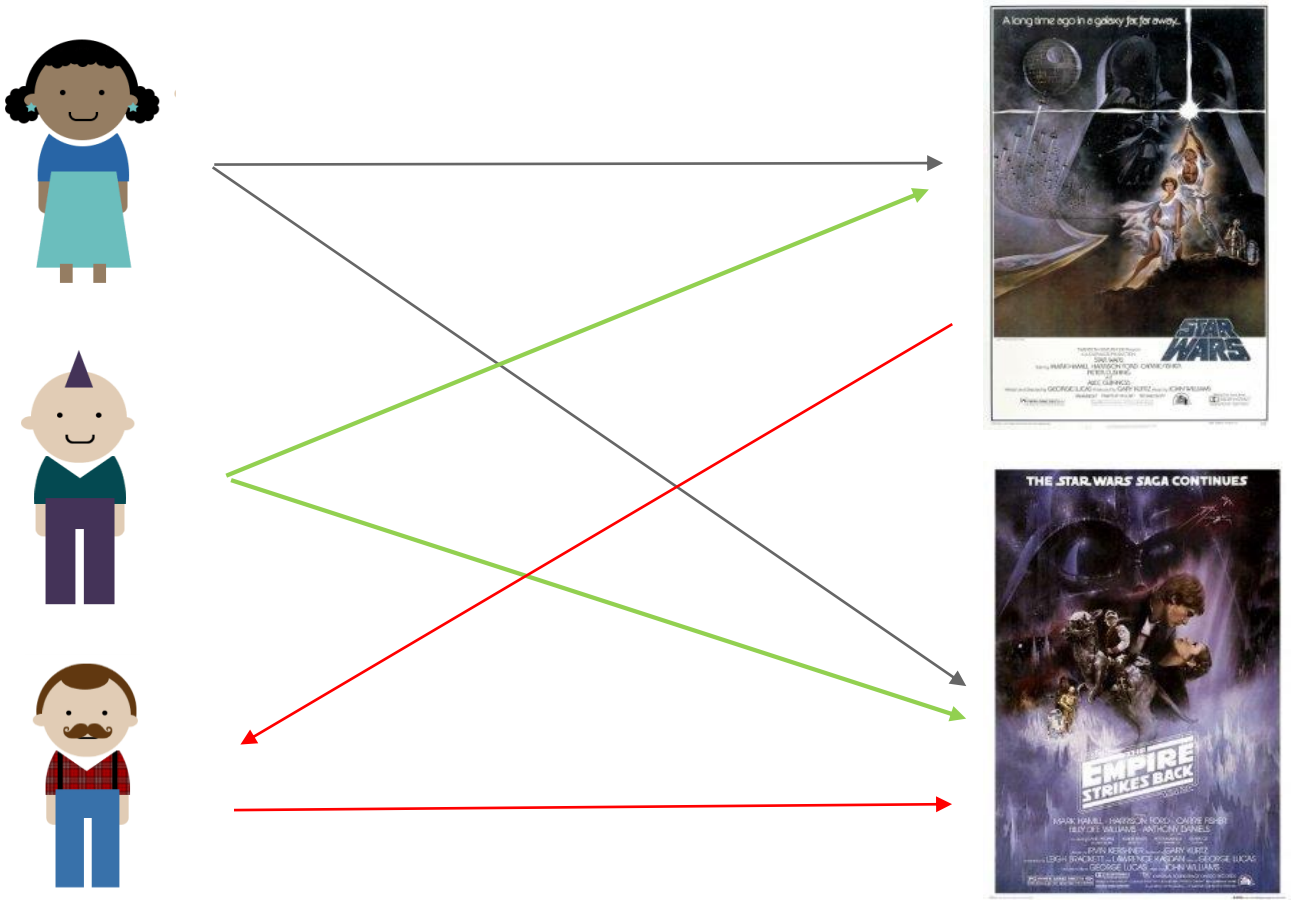
Item-Based Collaborative Filtering



Item-Based Collaborative Filtering



Item-Based Collaborative Filtering



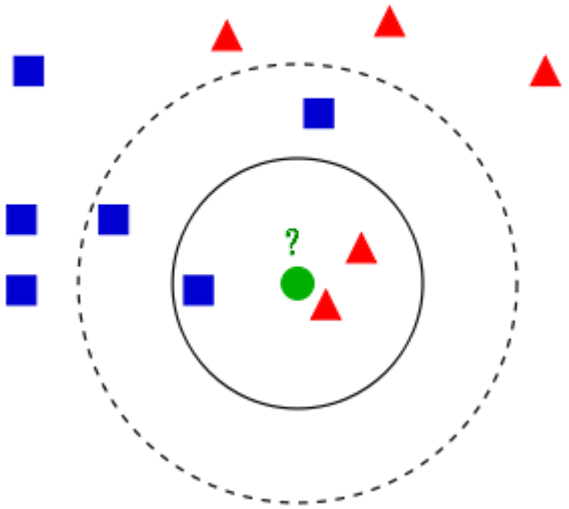
Let’s Do This

- Next, we’ll use Python to create real “movie similarities” using the real MovieLens data set.
  - In addition to being important for item-based collaborative filtering, these results are valuable in themselves – think “people who liked X also liked Y”
- It’s real world data, and we’ll encounter real world problems
- Then we’ll use those results to create movie recommendations for individuals

K-Nearest Neighbor

K-Nearest Neighbor (KNN)

- Used to classify new data points based on “distance” to known data
- Find the K nearest neighbors, based on your distance metric
- Let them all vote on the classification
- That’s it!



It’s Really That Simple

- Although it’s one of the simplest machine learning models there is – it still qualifies as “supervised learning”.
- But let’s do something more complex with it
- Movie similarities just based on metadata!

Customers Who Watched This Item Also Watched

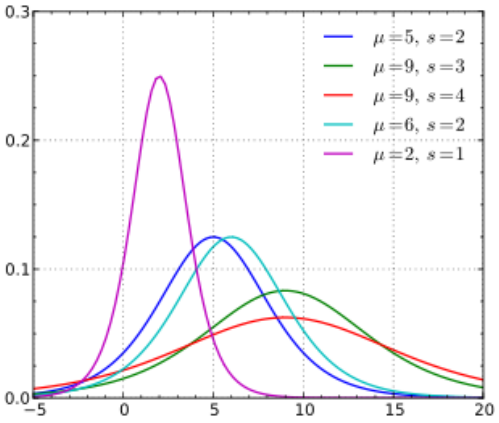
Discrete Choice Models

Discrete Choice Models

- Predict some choice people have between discrete alternatives
  - Do I take the train, bus, or car to work today? (Multinomial choice)
  - Which college will I go to? (Multinomial)
  - Will I cheat on my spouse? (Binary)
- The alternatives must be finite, exhaustive, and mutually exclusive

Discrete Choice Models

- Use some sort of regression on the relevant attributes
  - Attributes of the people
  - Variables of the alternatives
- Generally uses Logit or Probit models
  - Logistic Regression, Probit Model
  - Based on some utility function you define
  - Similar – one uses logistic distribution, Probit uses normal distribution. Logistic looks a lot like normal, but with fatter tails (higher kurtosis)



Example

- Will my spouse cheat on me?

The Curse of Dimensionality

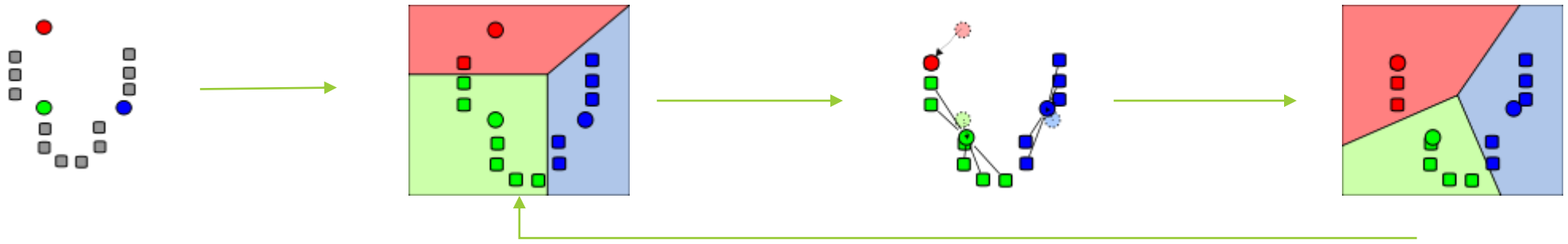
Principal Component Analysis

What is the curse of dimensionality?

- Many problems can be thought of as having a huge number of “dimesions”
- For example, in recommending movies, the ratings vector for each movie may represent a dimension – every movie is its own dimension!
- That makes your head hurt. It’s tough to visualize.
- *Dimensionality reduction* attempts to distill higher-dimensional data down to a smaller number of dimensions, while preserving as much of the variance in the data as possible.

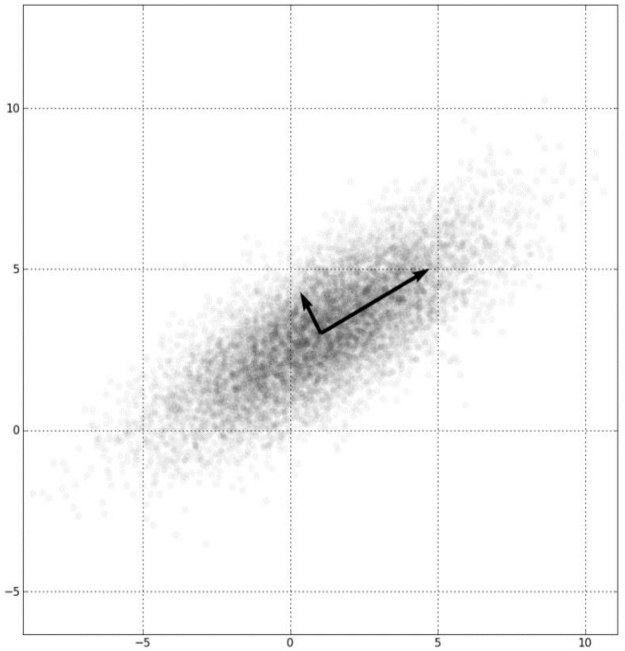
Remember K-Means Clustering?

- This is an example of a dimensionality reduction algorithm.
- It reduces data down to K dimensions.



Another way: Principal Component Analysis (PCA)

- Involves fancy math – but at a high level:
- Finds “eigenvectors” in the higher dimensional data
  - These define hyperplanes that split the data while preserving the most variance in it
  - The data gets projected onto these hyperplanes, which represent the lower dimensions you want to represent
  - A popular implementation of this is called Singular Value Decomposition (SVD)
- Also really useful for things like image compression and facial recognition



Example: Visualizing 4-D Iris Flower Data

- The “Iris dataset” comes with scikit-learn
- An Iris flower has petals and sepals (the lower, supportive part of the flower.)
- We know the length and width of the petals and sepals for many Iris specimens
  - That’s four dimensions! Ow.
  - We also know the subspecies classification of each flower
- PCA lets us visualize this in 2 dimensions instead of 4, while still preserving variance.

Example: Visualizing 4-D Iris Flower Data

- Yet again, scikit-learn makes this complex technique really easy.

ETL and ELT

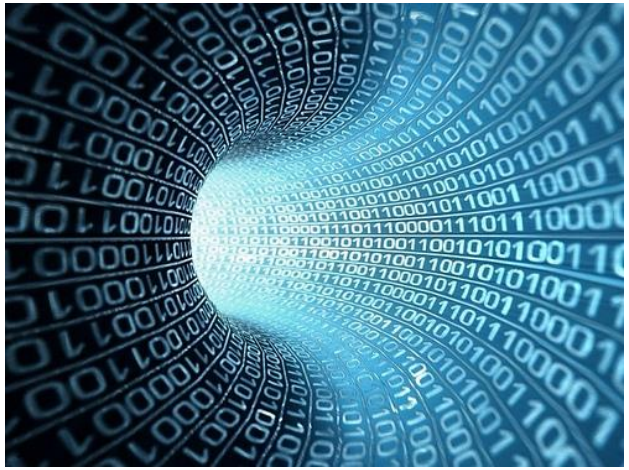
Data Warehousing Introduction

What is Data Warehousing?

- A large, centralized database that contains information from many sources
- Often used for business analysis in large corporations or organizations
- Queried via SQL or tools (i.e. Tableau)
- Often entire departments are dedicated to maintaining a data warehouse
  - Data normalization is tricky – how does all of this data relate to each other? What views do people need?
  - Maintaining the data feeds is a lot of work
  - Scaling is tricky

ETL: Extract, Transform, Load

- ETL and ELT refer to how data gets into a data warehouse.
- Traditionally, the flow was Extract, Transform, Load:
  - Raw data from operational systems is first periodically *extracted*
  - Then, the data is *transformed* into the schema needed by the DW
  - Finally, the data is *load*ed into the data warehouse, already in the structure needed
- But what if we're dealing with "big data"? That transform step can turn into a big problem.



ELT: Extract, Load, Transform

- Today, a huge Oracle instance isn't the only choice for a large data warehouse
- Things like Hive let you host massive databases on a Hadoop cluster
- Or, you might store it in a large, distributed NoSQL data store
  - …and query it using things like Spark or MapReduce
- The scalability of Hadoop lets you flip the loading process on its head
  - Extract raw data as before
  - Load it in as-is
  - Then use the power of Hadoop to transform it in-place

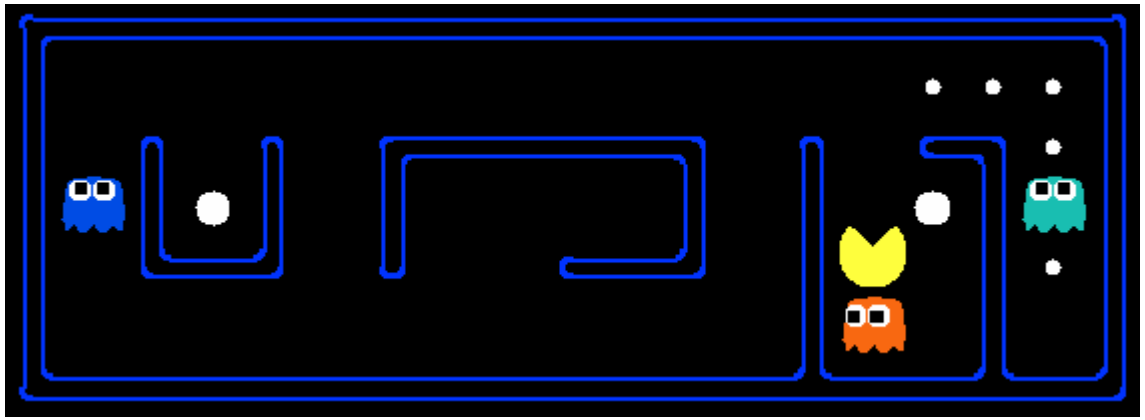
Lots more to explore

- Data warehousing is a discipline in itself, too big too cover here
- Check out other courses on Big Data, Spark, and MapReduce
  - We will cover Spark in more depth later in this course.

Reinforcement Learning

Reinforcement Learning

- You have some sort of agent that “explores” some space
- As it goes, it learns the value of different state changes in different conditions
- Those values inform subsequent behavior of the agent
- Examples: Pac-Man, Cat & Mouse game
- Yields fast on-line performance once the space has been explored



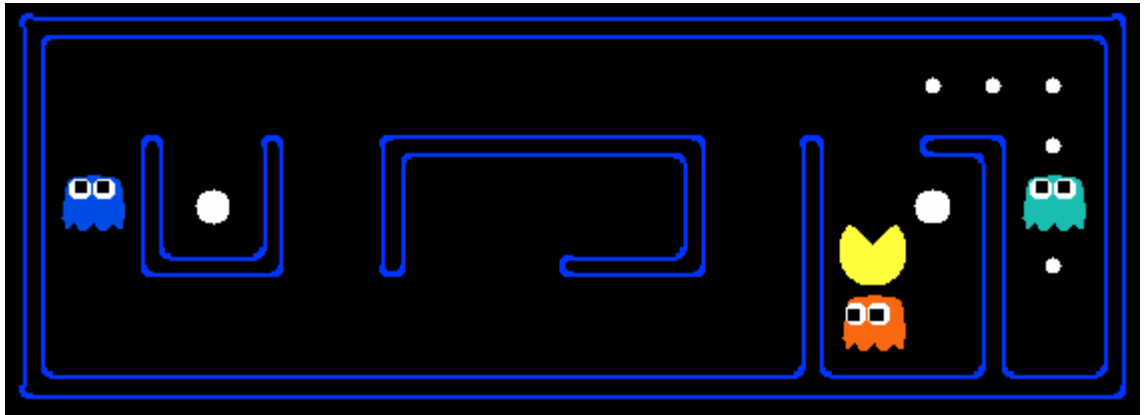
Q-Learning

- A specific implementation of reinforcement learning
- You have:
  - A set of environmental states *s*
  - A set of possible actions in those states *a*
  - A value of each state/action Q
- Start off with Q values of 0
- Explore the space
- As bad things happen after a given state/action, reduce its Q
- As rewards happen after a given state/action, increase its Q

Q(s, a) += alpha * (reward(s,a) + max(Q(s') - Q(s,a)) where s is the previous state, a is the previous action,s' is the current state, and alpha is the discount factor (set to .5 here).

Q-Learning

- What are some state/actions here?
  - Pac-man has a wall to the West
  - Pac-man dies if he moves one step South
  - Pac-man just continues to live if going North or East
- You can “look ahead” more than one step by using a discount factor when computing Q (here s is previous state, s’ is current state)
  - Q(s,a) += discount * (reward(s,a) + max(Q(s’)) – Q(s,a))



The exploration problem

- How do we efficiently explore all of the possible states?
  - Simple approach: always choose the action for a given state with the highest Q. If there's a tie, choose at random
    - But that's really inefficient, and you might miss a lot of paths that way
  - Better way: introduce an epsilon term
    - If a random number is less than epsilon, don't follow the highest Q, but choose at random
    - That way, exploration never totally stops
    - Choosing epsilon can be tricky

Fancy Words

- Markov Decision Process
  - From Wikipedia: **Markov decision processes (MDPs)** provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.
  - Sound familiar? MDP's are just a way to describe what we just did using mathematical notation.
  - States are still described as s and s'
  - State transition functions are described as $P_a(s, s')$
  - Our "Q" values are described as a reward function $R_a(s, s')$
- Even fancier words! An MDP is a *discrete time stochastic control process.*

More Fancy Words

- Dynamic Programming
  - From Wikipedia: **dynamic programming** is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions - ideally, using a memory-based data structure. The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in storage space.
  - Sound familiar?

So to recap

- You can make an intelligent Pac-Man in a few steps:
  - Have it semi-randomly explore different choices of movement (actions) given different conditions (states)
  - Keep track of the reward or penalty associated with each choice for a given state/action (Q)
  - Use those stored Q values to inform its future choices
- Pretty simple concept. But hey, now you can say you understand reinforcement learning, Q-learning, Markov Decision Processes, and Dynamic Programming!

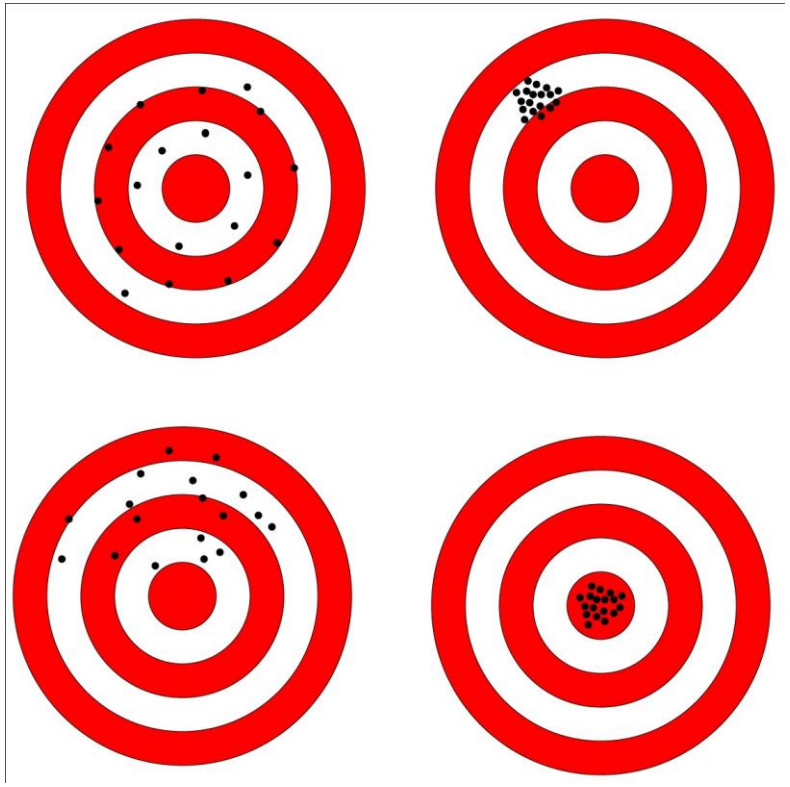
Implementing Reinforcement Learning

- Python Markov Decision Process Toolbox:
  - http://pymdptoolbox.readthedocs.org/en/latest/api/mdp.html
- Cat & Mouse Example:
  - https://github.com/studywolf/blog/tree/master/RL/Cat%20vs%20Mouse%20exploration
- Pac-Man Example:
  - https://inst.eecs.berkeley.edu/~cs188/sp12/projects/reinforcement/reinforcement.html

The Bias / Variance Tradeoff

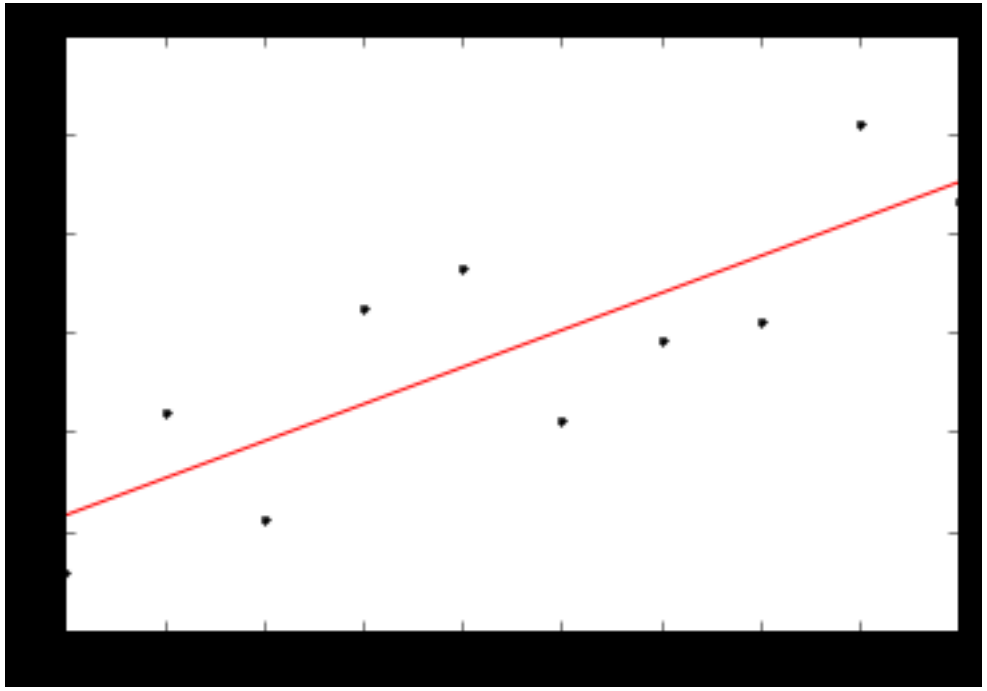
Bias and Variance

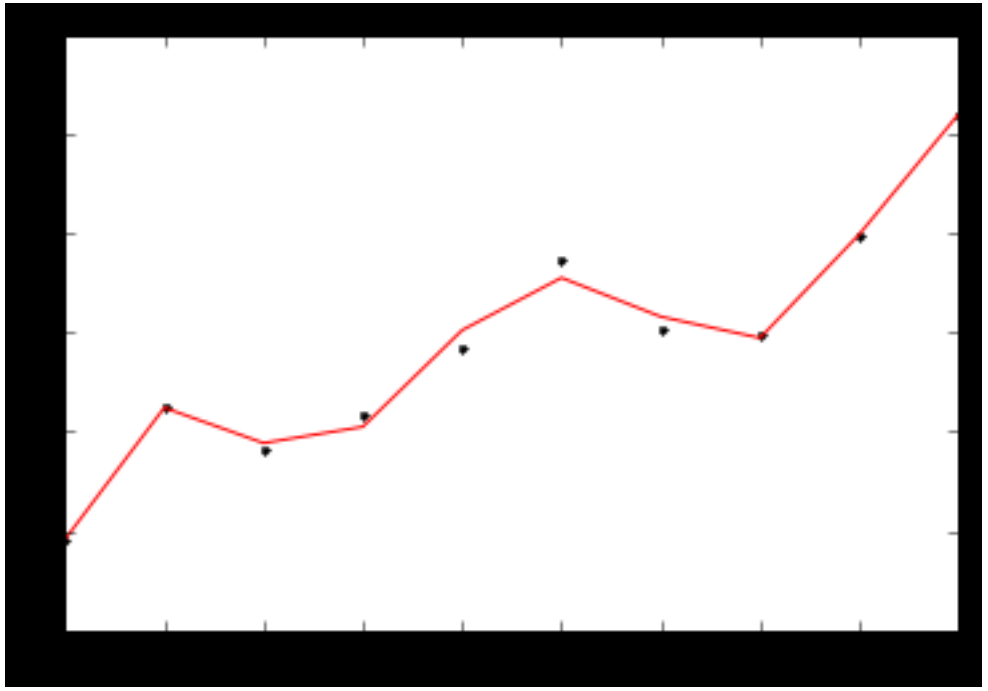
- *Bias* is how far removed the mean of your predicted values is from the “real” answer
- *Variance* is how scattered your predicted values are from the “real” answer
- Describe the bias and variance of these four cases (assuming the center is the correct result)



Often you need to choose between bias and variance

- It comes down to overfitting vs underfitting your data





But what you really care about is error

- Bias and variance both contribute to error
  - $Error = Bias^2 + Variance$
- But it's error you want to minimize, not bias or variance specifically
- A complex model will have high variance and low bias
- A too-simple model will have low variance and high bias
- But both may have the same error – the optimal complexity is in the middle

Tying it to earlier lessons

- Increasing K in K-Nearest-Neighbors decreases variance and increases bias (by averaging together more neighbors)
- A single decision tree is prone to overfitting – high variance
  - But a random forest decreases that variance.

Avoiding Overfitting

Using K-Fold Cross Validation

Review: K-Fold Cross Validation

- One way to further protect against overfitting is *K-fold cross validation*
- Sounds complicated. But it's a simple idea:
  - Split your data into K randomly-assigned segments
  - Reserve one segment as your test data
  - Train on each of the remaining K-1 segments and measure their performance against the test set
  - Take the average of the K-1 r-squared scores
- Prevents you from overfitting to a single train/test split

Using K-Fold Cross Validation

- Scikit-learn makes this really easy. Even easier than just a single train/test split.
- In practice, you need to try different variations of your model and measure the mean accuracy using K-Fold Cross validation until you find a sweet spot

Let’s Play

- Use K-Fold Cross Validation with a SVC model of Iris classification.
  We’ll see that without K-Fold, we could overfit the model.

Cleaning Your Data

Cleaning your Data

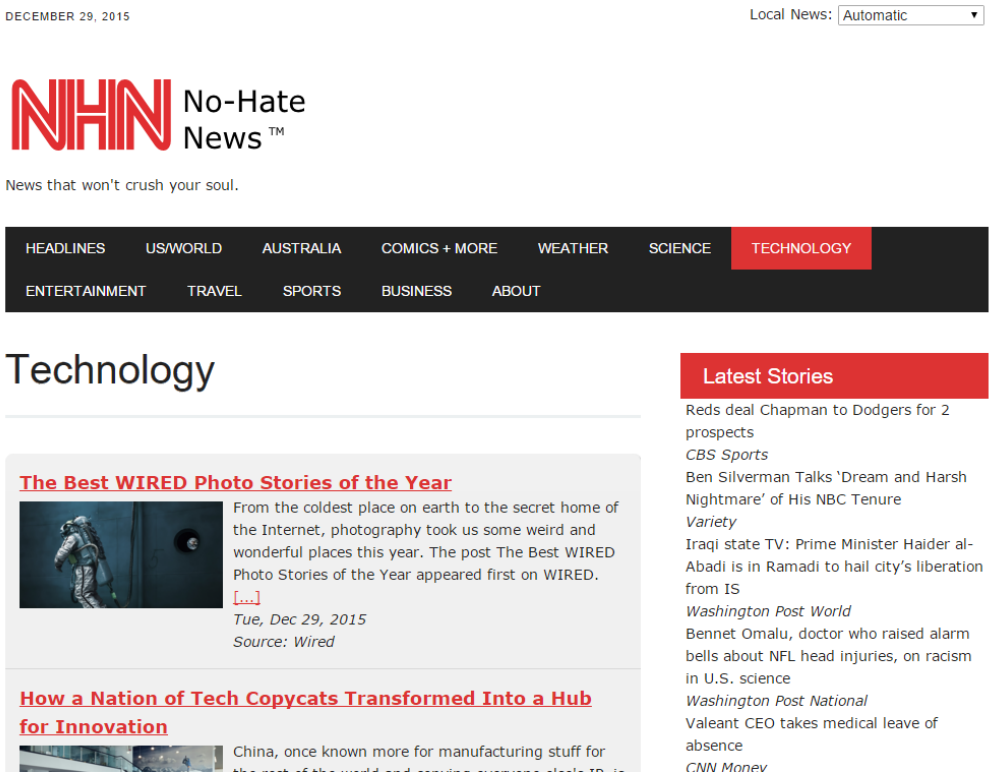
- The reality is, much of your time as a data scientist will be spent preparing and “cleaning” your data
  - Outliers
  - Missing Data
  - Malicious Data
  - Erroneous Data
  - Irrelevant Data
  - Inconsistent Data
  - Formatting

Garbage In, Garbage Out

- Look at your data! Examine it!
- Question your results!
  - And *always* do this – not just when you don't get a result that you like!

Let’s analyze some web log data.

- All I want is the most-popular pages on my non-profit news website.
- How hard can that be?



Normalizing Numerical Data

The importance of normalizing data

- If your model is based on several numerical attributes – are they comparable?
  - Example: ages may range from 0-100, and incomes from 0-billions
  - Some models may not perform well when different attributes are on very different scales
  - It can result in some attributes counting more than others
  - Bias in the attributes can also be a problem.

Examples

- Scikit-learn's PCA implementation has a "whiten" option that does this for you. Use it.
- Scikit-learn has a preprocessing module with handy normalize and scale functions
- Your data may have "yes" and "no" that needs to be converted to "1" and "0"

Read the docs

- Most data mining and machine learning techniques work fine with raw, un-normalized data
- But double check the one you're using before you start.
- Don't forget to re-scale your results when you're done!

Dealing with Outliers

Dealing with Outliers

- Sometimes it's appropriate to remove outliers from your training data
- Do this responsibly! Understand why you are doing this.
- For example: in collaborative filtering, a single user who rates thousands of movies could have a big effect on everyone else's ratings. That may not be desirable.
- Another example: in web log data, outliers may represent bots or other agents that should be discarded.
- But if someone really wants the mean income of US citizens for example, don't toss out Donald Trump just because you want to.

Dealing with Outliers

- Our old friend standard deviation provides a principled way to classify outliers.
- Find data points more than some multiple of a standard deviation in your training data.
- What multiple? You just have to use common sense.

Let’s play with some data.

Installing Apache Spark on Windows

Installing Spark on Windows

- Install a JDK
- Install Python (but you should already have this)
- Install a pre-built version of Spark for Hadoop
- Create a conf/log4j.properties file to change the warning level
- Add a SPARK_HOME environment variable
- Add %SPARK_HOME%\bin to your PATH
- Set HADOOP_HOME to c:\winutils
- Install winutils.exe to c:\winutils\bin

Installing Spark on other OS's

- Pretty much the same, but look up how to set environment variables on your OS
- Winutils.exe not needed of course

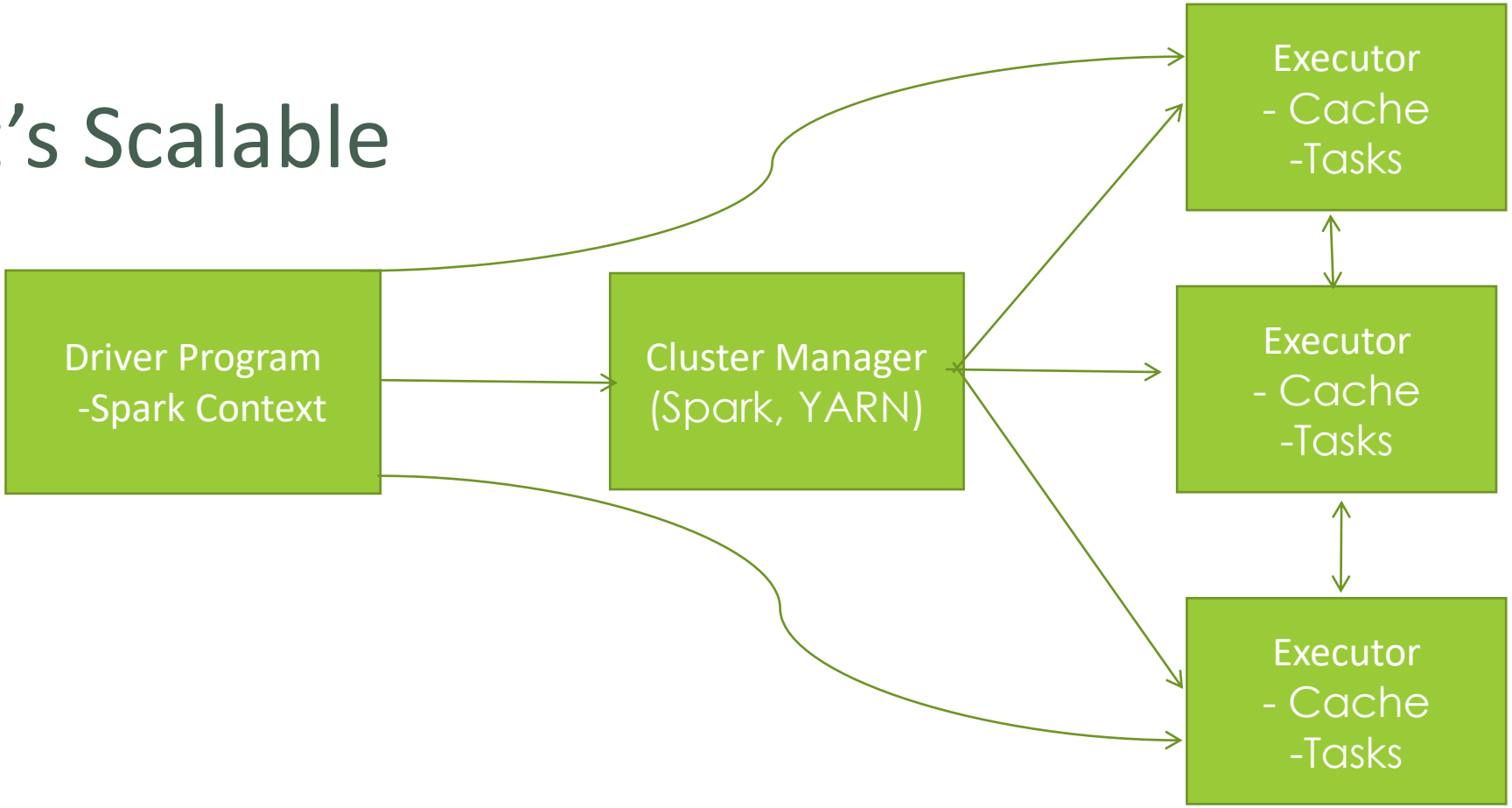
Let’s Do It

Spark Introduction

What is Spark?

- "A fast and general engine for large-scale data processing"

It’s Scalable



It’s Fast

- "Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk."
- DAG Engine (directed acyclic graph) optimizes workflows

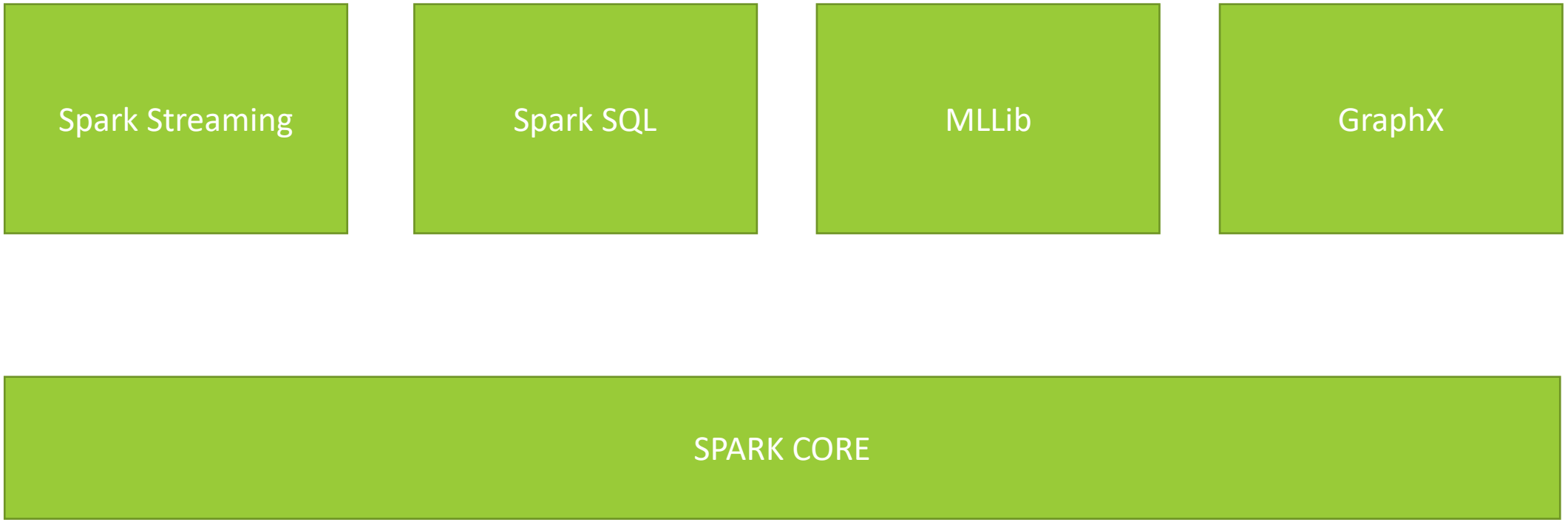
It’s Hot

- Amazon
- Ebay: log analysis and aggregation
- NASA JPL: Deep Space Network
- Groupon
- TripAdviser
- Yahoo
- Many others: https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark

It’s Not That Hard

- Code in Python, Java, or Scala
- Built around one main concept: the Resilient Distributed Dataset (RDD)

Components of Spark



Python vs. Scala

- Why Python?
  - No need to compile, manage dependencies, etc.
  - Less coding overhead
  - You already know Python
  - Lets us focus on the concepts instead of a new language
- But...
  - Scala is probably a more popular choice with Spark.
  - Spark is built in Scala, so coding in Scala is "native" to Spark
  - New features, libraries tend to be Scala-first.

Fear Not

- Python and Scala look very similar in Spark.

Python code to square numbers in a data set:

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
```

Scala code to square numbers in a data set:

```
val nums = sc.parallelize(List(1, 2, 3, 4))
val squared = nums.map(x => x * x).collect()
```

Resilient Distributed Datasets (RDDs)

RDD

- Resilient
- Distributed
- Dataset

The SparkContext

- Created by your driver program
- Is responsible for making RDD's resilient and distributed!
- Creates RDD's
- The Spark shell creates a "sc" object for you

Creating RDD's

- nums = parallelize([1, 2, 3, 4])
- sc.textFile("file:///c:/users/frank/gobs-o-text.txt")
  - or s3n:// , hdfs://
- hiveCtx = HiveContext(sc) rows = hiveCtx.sql("SELECT name, age FROM users")
- Can also create from:
  - JDBC
  - Cassandra
  - HBase
  - Elastisearch
  - JSON, CSV, sequence files, object files, various compressed formats

Transforming RDD's

- map
- flatmap
- filter
- distinct
- sample
- union, intersection, subtract, cartesian

Map() example

- rdd = sc.parallelize([1, 2, 3, 4])
- rdd.map(lambda x: x*x)

- This yields 1, 4, 9, 16

What's that lambda thing?

- Many RDD methods accept a *function* as a parameter
- rdd.map(lambda x: x*x)
- Is the same thing as
- def squareIt(x):
- 	return x*x
- rdd.map(squareIt)
- There, you now understand functional programming.

RDD Actions

- collect
- count
- countByValue
- take
- top
- reduce
- … and more ...

Lazy Evaluation

- Nothing actually happens in your driver program until an action is called!

Introducing MLLib

Some MLLib Capabilities

- Feature extraction
  - Term Frequency / Inverse Document Frequency useful for search
- Basic statistics
  - Chi-squared test, Pearson or Spearman correlation, min, max, mean, variance
- Linear regression, logistic regression
- Support Vector Machines
- Naïve Bayes classifier
- Decision trees
- K-Means clustering
- Principal component analysis, singular value decomposition
- Recommendations using Alternating Least Squares

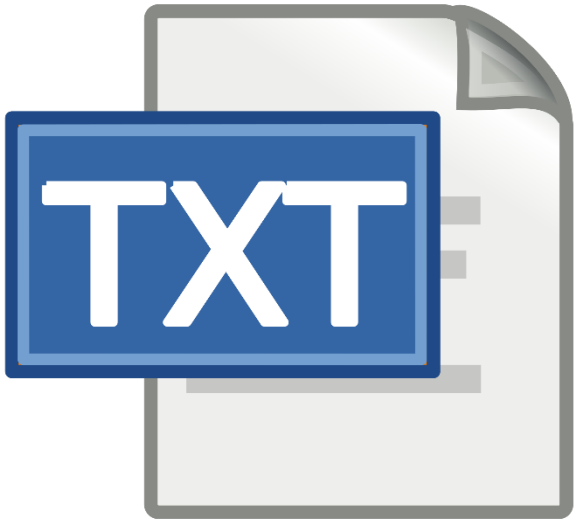
Special MLLib Data Types

- Vector (dense or sparse)
- LabeledPoint
- Rating

TF-IDF

TF-IDF

- Stands for *Term Frequency* and *Inverse Document Frequency*
- Important data for search – figures out what terms are most relevant for a document
- Sounds fancy!



TF-IDF Explained

- *Term Frequency* just measures how often a word occurs in a document
  - A word that occurs frequently is probably important to that document's meaning
- *Document Frequency* is how often a word occurs in an entire set of documents, i.e., all of Wikipedia or every web page
  - This tells us about common words that just appear everywhere no matter what the topic, like "a", "the", "and", etc.

TF-IDF Explained

- So a measure of the relevancy of a word to a document might be:

$$\frac{Term\ Frequency}{Document\ Frequency}$$

Or: Term Frequency * Inverse Document Frequency

That is, take how often the word appears in a document, over how often it just appears everywhere. That gives you a measure of how important and unique this word is for this document

TF-IDF In Practice

- We actually use the log of the IDF, since word frequencies are distributed exponentially. That gives us a better weighting of a words overall popularity
- TF-IDF assumes a document is just a “bag of words”
  - Parsing documents into a bag of words can be most of the work
  - Words can be represented as a hash value (number) for efficiency
  - What about synonyms? Various tenses? Abbreviations? Capitalizations? Misspellings?
- Doing this at scale is the hard part
  - That’s where Spark comes in!

Using TF-IDF

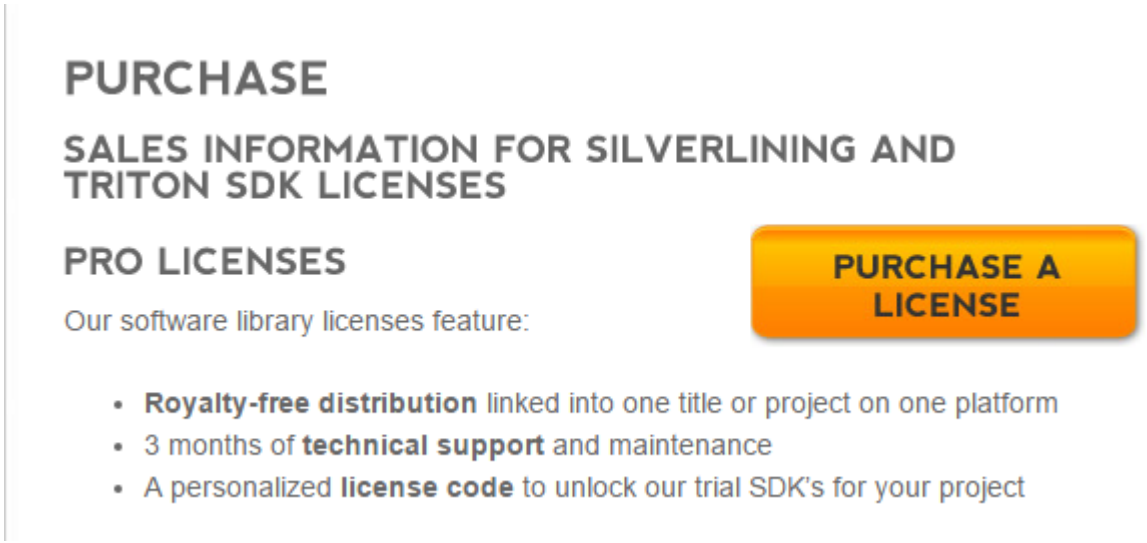
- A very simple search algorithm could be:
  - Compute TF-IDF for every word in a corpus
  - For a given search word, sort the documents by their TF-IDF score for that word
  - Display the results

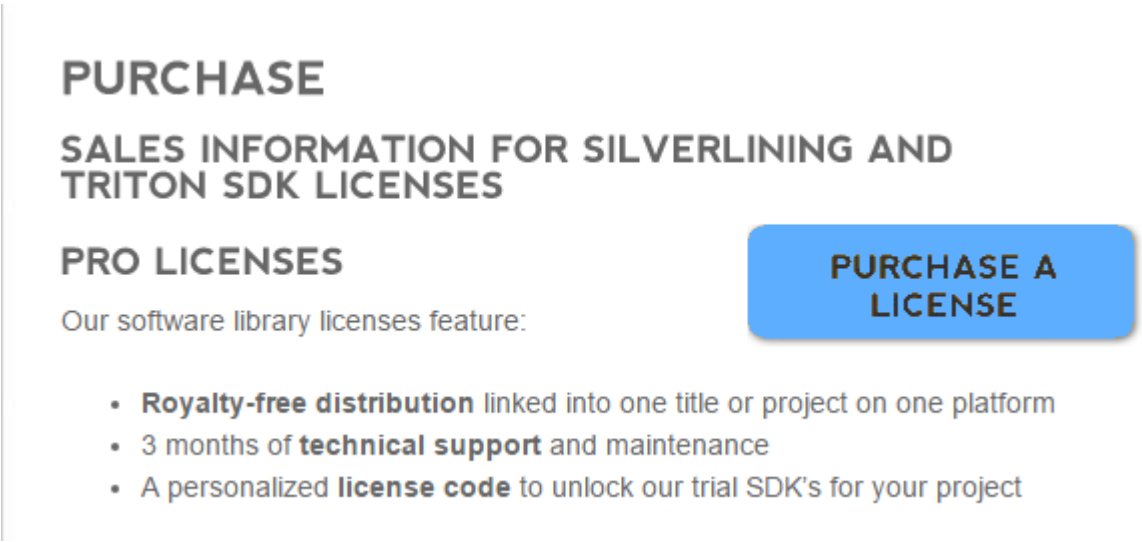
Let’s use TF-IDF on Wikipedia

A/B Tests

What is an A/B test?

- A controlled experiment, usually in the context of a website
- You test the performance of some change to your website (the variant) and measure conversion relative to your unchanged site (the control.)





What sorts of things can you test?

- Design changes
- UI flow
- Algorithmic changes
- Pricing changes
- You name it

How do you measure conversion

- Ideally choose what you are trying to influence
  - Order amounts
  - Profit
  - Ad clicks
  - Order quantity
- But attributing actions downstream from your change can be hard
  - Especially if you’re running more than one experiment

Variance is your Enemy

- Common mistake:
  - Run a test for some small period of time that results in a few purchases to analyze
  - You take the mean order amount from A and B, and declare victory or defeat
  - But, there's so much random variation in order amounts to begin with, that your result was just based on chance.
  - You then fool yourself into thinking some change to your website, which could actually be harmful, has made tons of money.

Variance is your Enemy

- Sometimes you need to also look at conversion metrics with less variance
- Order quantities vs. order dollar amounts, for example

T-Tests and P-Values

Determining significance

- So, how do we know if a result is likely to be “real” as opposed to just random variation?
- T-tests and P-values

The T-Statistic

- A measure of the difference between the two sets expressed in units of standard error
- The size of the difference relative to the variance in the data
- A high t value means there's probably a real difference between the two sets
- Assumes a normal distribution of behavior
  - This is a good assumption if you're measuring revenue as conversion
  - See also: Fisher's exact test (for clickthrough rates), E-test (for transactions per user) and chi-squared test (for product quantities purchased)

The P-Value

- Think of it as the probability of A and B satisfying the “null hypothesis”
- So, a low P-Value implies significance.
- It is the probability of an observation lying at an extreme t-value assuming the null hypothesis

Using P-values

- Choose some threshold for “significance” before your experiment
  - 1%? 5%?
- When your experiment is over:
  - Measure your P-value
  - If it’s less than your significance threshold, then you can reject the null hypothesis
    - If it’s a positive change, roll it out
    - If it’s a negative change, discard it before you lose more money.

Let’s work through an example.

How Long Do I Run an Experiment?

How do I know when I’m done with an A/B test?

- You have achieved significance (positive or negative)
- You no longer observe meaningful trends in your p-value
  - That is, you don’t see any indication that your experiment will “converge” on a result over time
- You reach some pre-established upper bound on time

A/B Test Gotchas

Correlation does not imply causation

- Even your low p-value score on a well-designed experiment does not imply causation!
  - It could still be random chance
  - Other factors could be at play
  - It’s your duty to ensure business owners understand this

Novelty Effects

- Changes to a website will catch the attention of previous users who are used to the way it used to be
  - They might click on something simply because it is new
  - But this attention won't last forever
- Good idea to re-run experiments much later and validate their impact
  - Often the "old" website will outperform the "new" one after awhile, simply because it is a change

Seasonal Effects

- An experiment run over a short period of time may only be valid for that period of time
  - Example: Consumer behavior near Christmas is very different than other times of year
  - An experiment run near Christmas may not represent behavior during the rest of the year

Selection Bias

- Sometimes your random selection of customers for A or B isn't really random
  - For example: assignment is based somehow on customer ID
  - But customers with low ID's are better customers than ones with high ID's
- Run an A/A test periodically to check
- Audit your segment assignment algorithms

Data Pollution

- Are robots (both self-identified and malicious) affecting your experiment?
  - Good reason to measure conversion based on something that requires spending real money
- More generally, are outliers skewing the result?

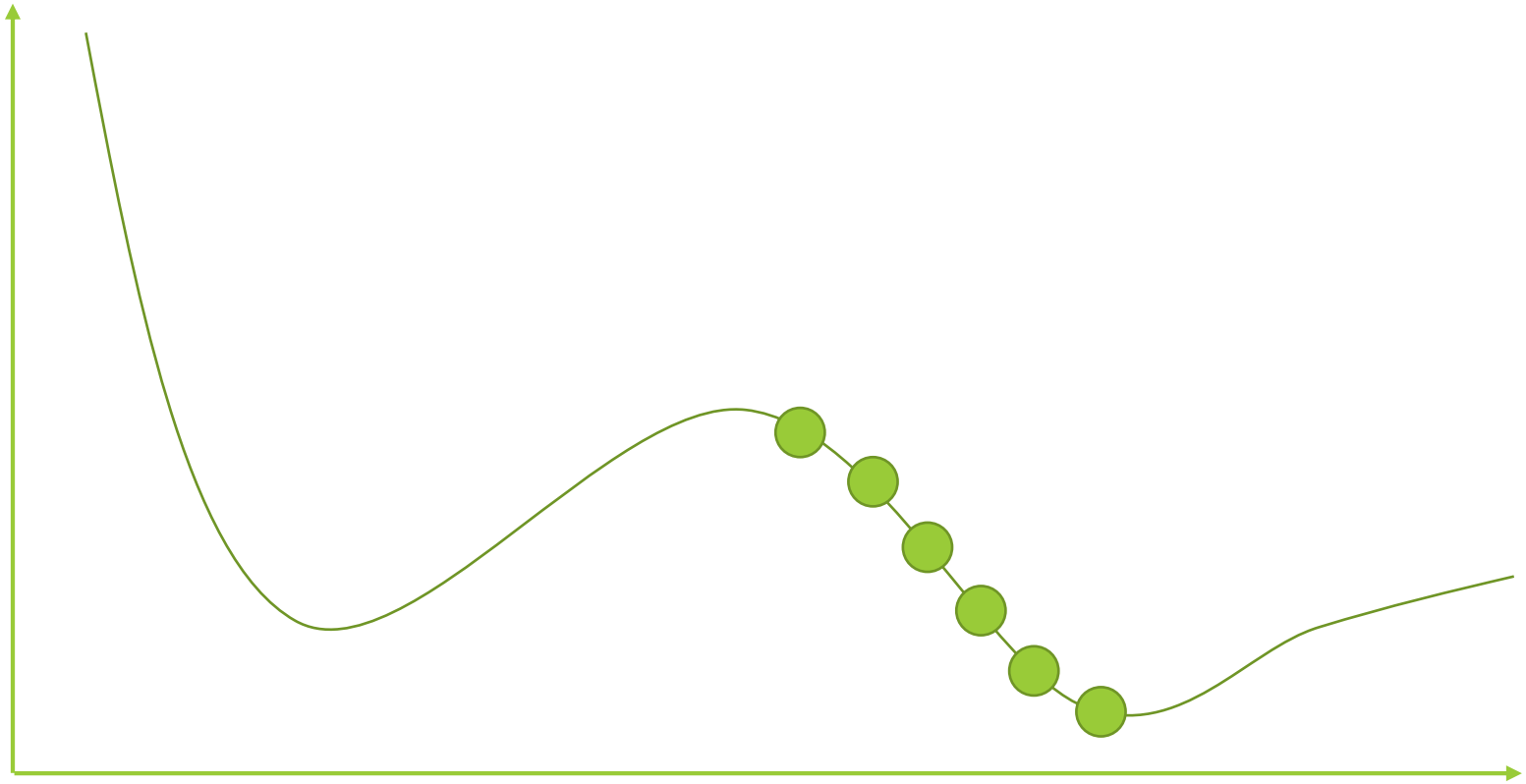
Attribution Errors

- Often there are errors in how conversion is attributed to an experiment
- Using a widely used A/B test platform can help mitigate that risk
  - If your is home-grown, it deserves auditing
- Watch for “gray areas”
  - Are you counting purchases toward an experiment within some given time-frame of exposure to it? Is that time too large?
  - Could other changes downstream from the change you’re measuring affect your results?
  - Are you running multiple experiments at once?

Deep Learning: Pre-Requisites

Understanding gradient descent, autodiff, and softmax

Gradient Descent



autodiff

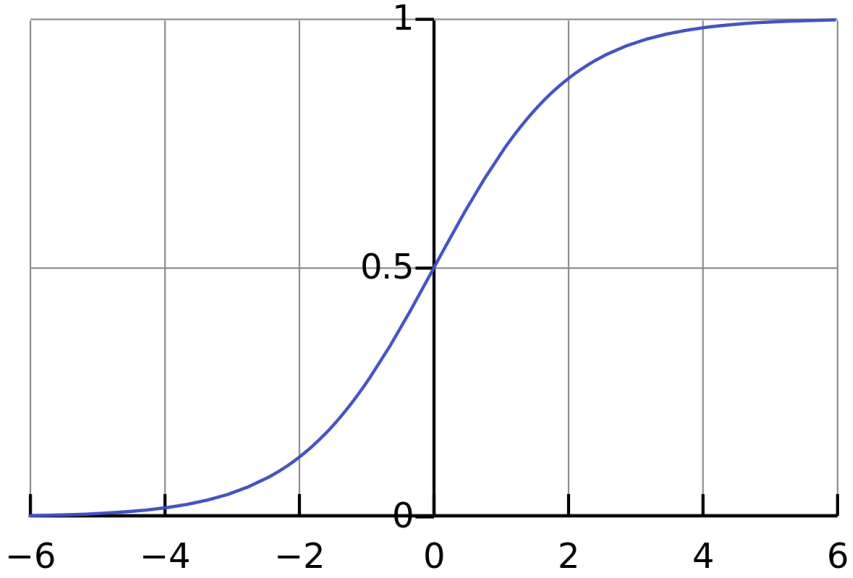
- Gradient descent requires knowledge of, well, the gradient from your cost function (MSE)
- Mathematically we need the first partial derivatives of all the inputs
  - This is hard and inefficient if you just throw calculus at the problem
- Reverse-mode autodiff to the rescue!
  - Optimized for many inputs + few outputs (like a neuron)
  - Computes all partial derivatives in # of outputs + 1 graph traversals
  - Still fundamentally a calculus trick – it's complicated but it works
  - This is what Tensorflow uses

softmax

- Used for classification
  - Given a score for each class
  - It produces a probability of each class
  - The class with the highest probability is the “answer” you get

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)},$$

x is a vector of input values
theta is a vector of weights



In review:

- Gradient descent is an algorithm for minimizing error over multiple steps
- Autodiff is a calculus trick for finding the gradients in gradient descent
- Softmax is a function for choosing the most probable classification given several input values

Introducing Artificial Neural Networks

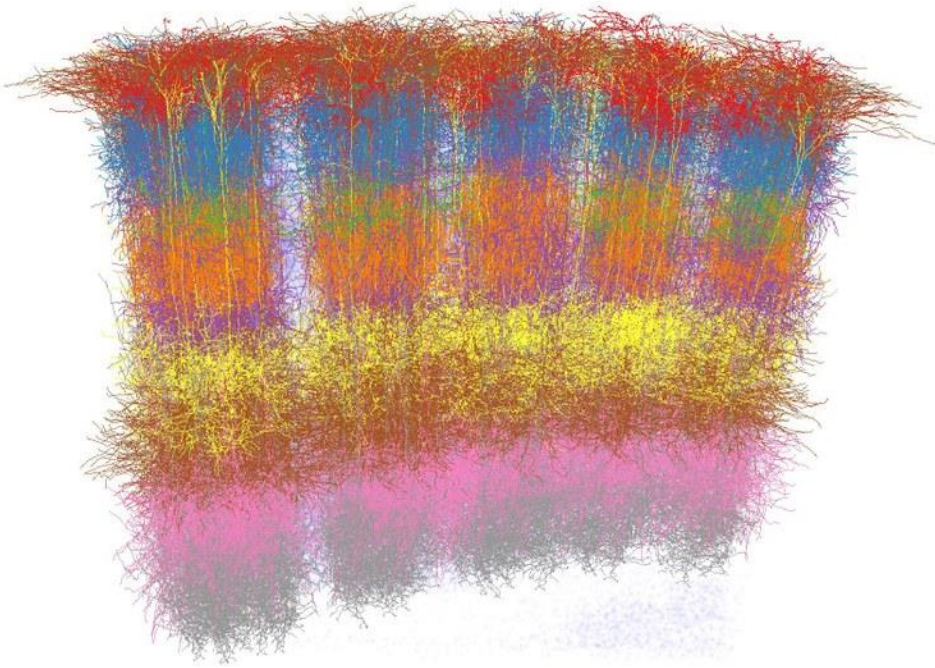
Evolving beyond nature

The biological inspiration

- Neurons in your cerebral cortex are connected via axons
- A neuron “fires” to the neurons it’s connected to, when enough of its input signals are activated.
- Very simple at the individual neuron level – but layers of neurons connected in this way can yield learning behavior.
- Billions of neurons, each with thousands of connections, yields a mind

Cortical columns

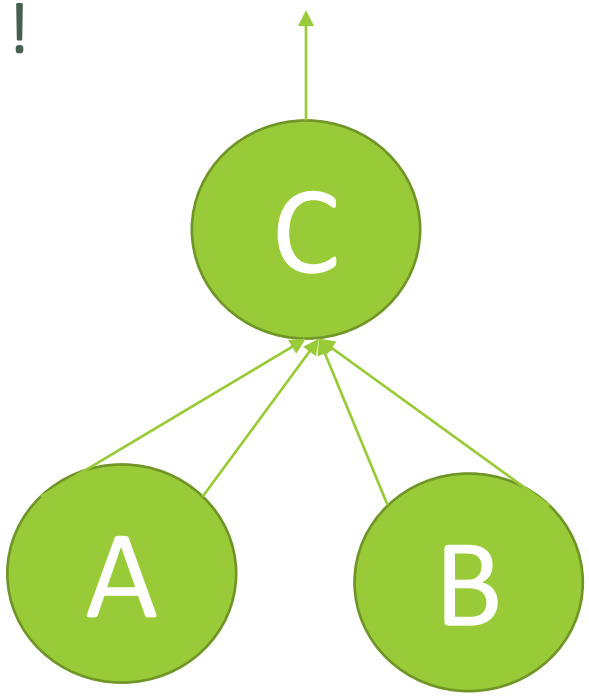
- Neurons in your cortex seem to be arranged into many stacks, or “columns” that process information in parallel
- “mini-columns” of around 100 neurons are organized into larger “hyper-columns”. There are 100 million mini-columns in your cortex
- This is coincidentally similar to how GPU’s work…



(credit: Marcel Oberlaender et al.)

The first artificial neurons

- 1943!!



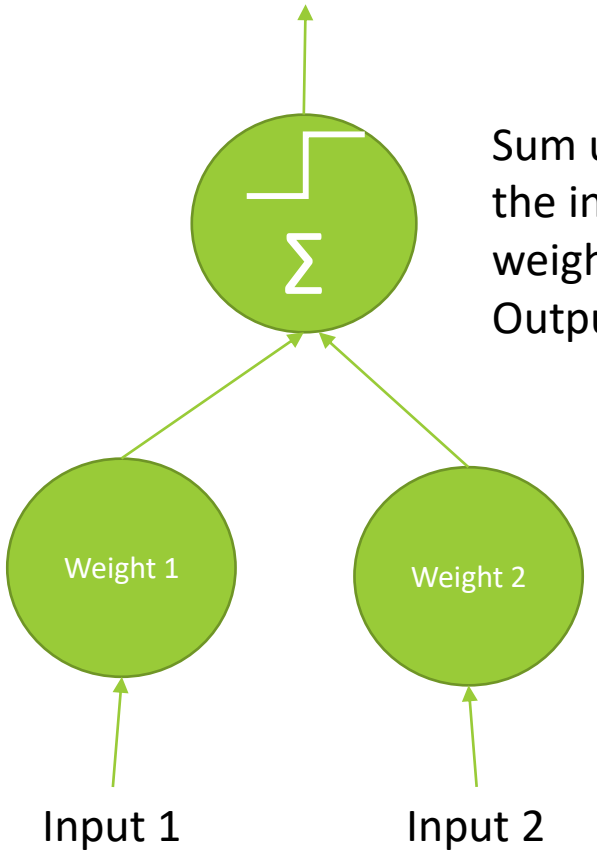
An artificial neuron “fires” if more than N input connections are active.

Depending on the number of connections from each input neuron, and whether a connection activates or suppresses a neuron, you can construct AND, OR, and NOT logical constructs this way.

This example would implement C = A OR B if the threshold is 2 inputs being active.

The Linear Threshold Unit (LTU)

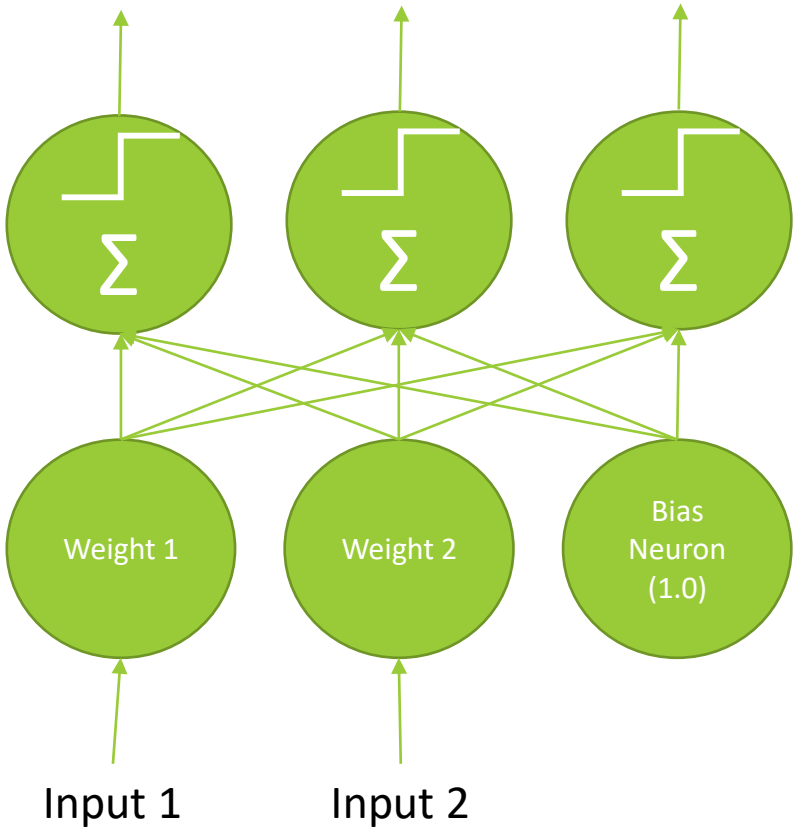
- 1957!
- Adds weights to the inputs; output is given by a step function



Sum up the products of the inputs and their weights
Output 1 if sum is >= 0

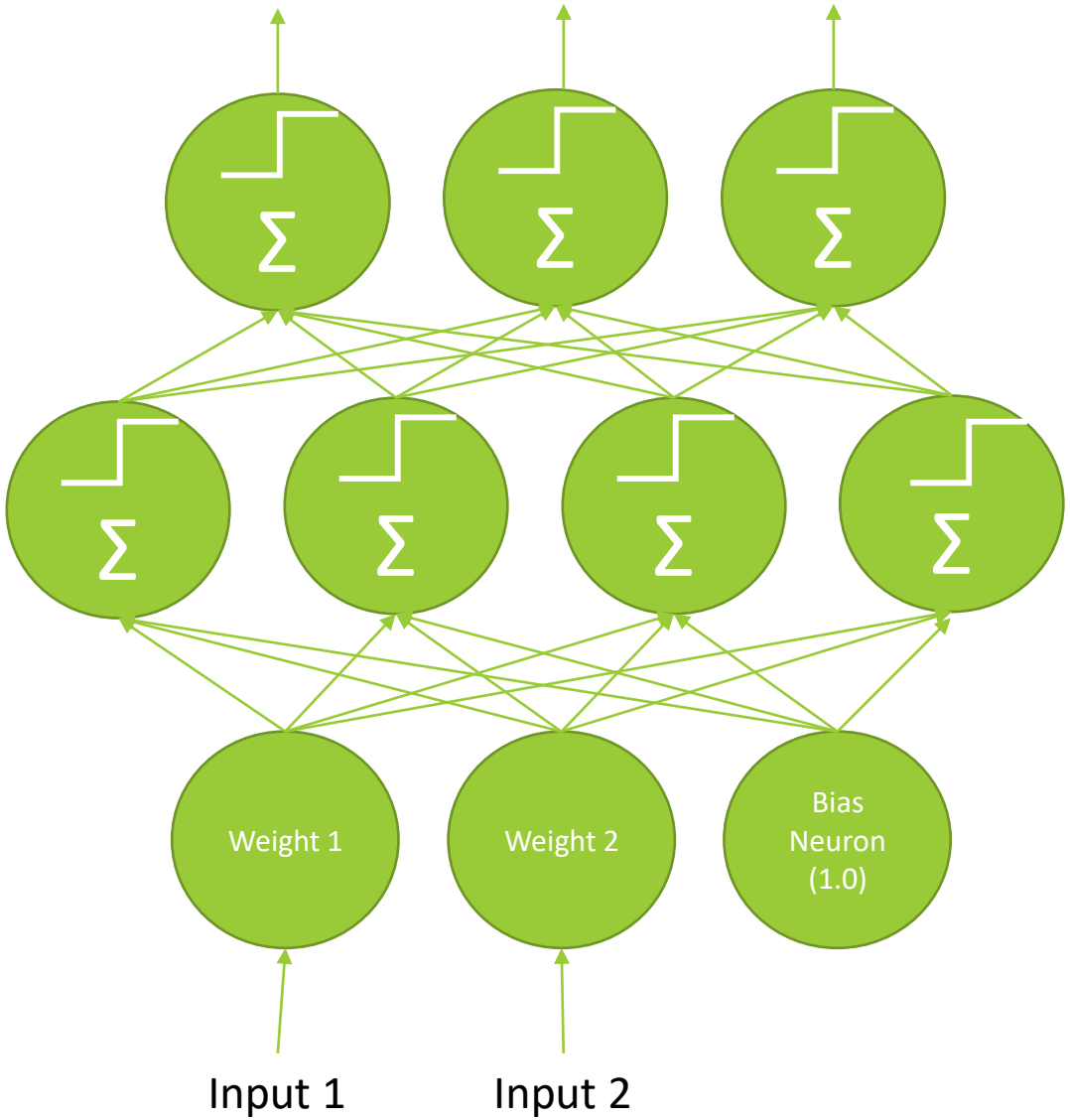
The Perceptron

- A layer of LTU's
- A perceptron can learn by reinforcing weights that lead to correct behavior during training
- This too has a biological basis ("cells that fire together, wire together")



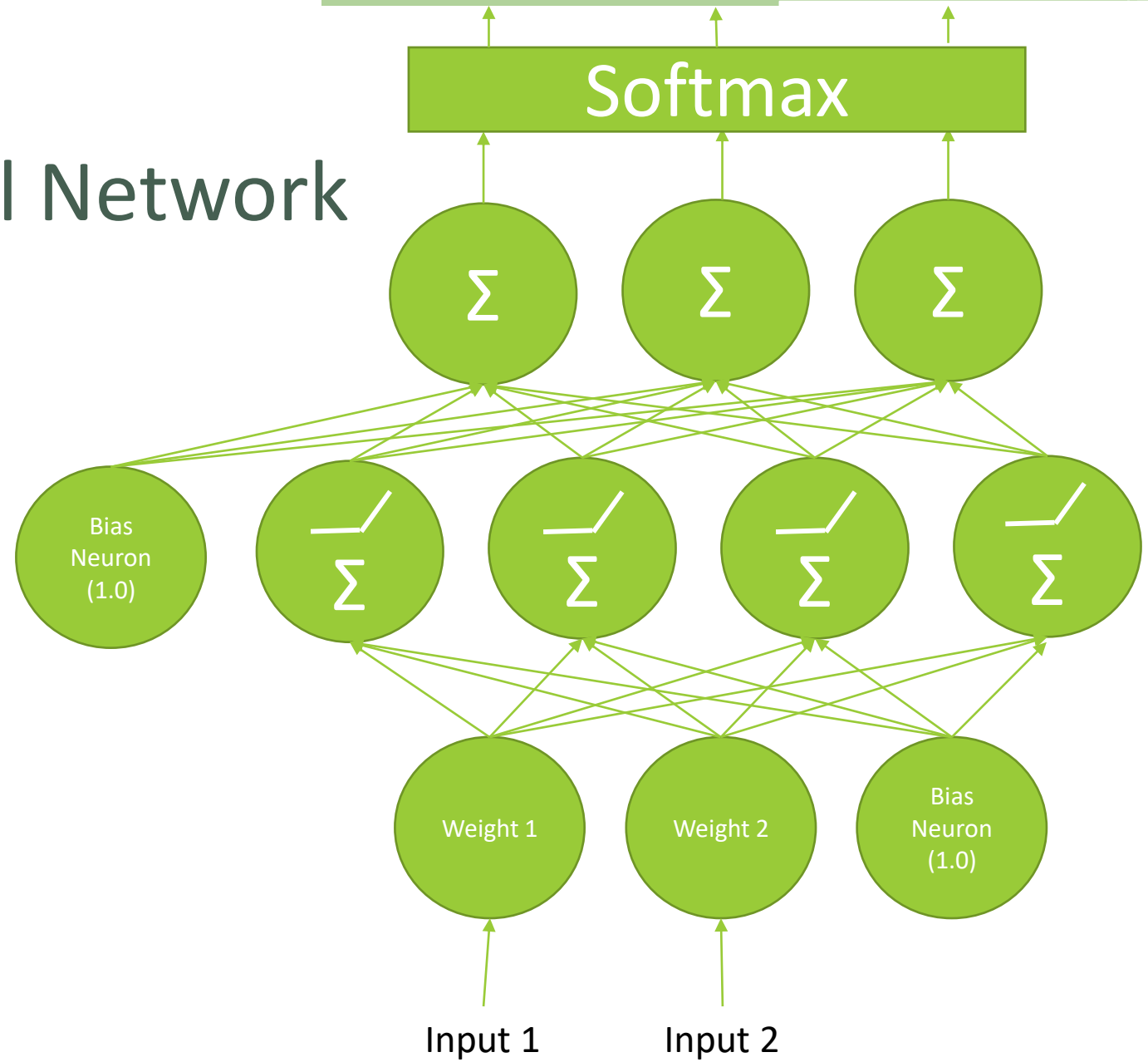
Multi-Layer Perceptrons

- Addition of “hidden layers”
- This is a Deep Neural Network
- Training them is trickier – but we’ll talk about that.



A Modern Deep Neural Network

- Replace step activation function with something better
- Apply softmax to the output
- Training using gradient descent



Let’s play

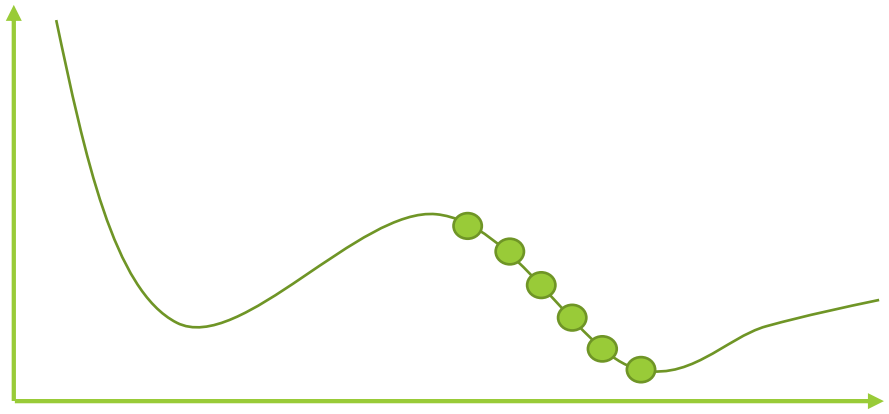
playground.tensorflow.org

Deep Learning

Constructing, training, and tuning multi-layer perceptrons

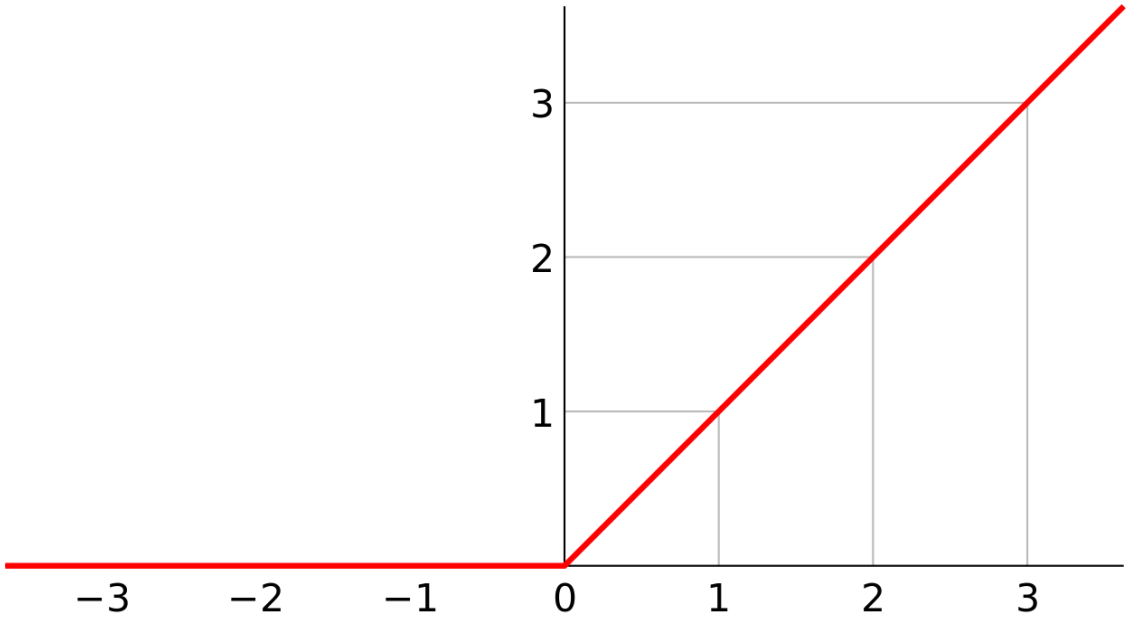
Backpropagation

- How do you train a MLP's weights? How does it **learn**?
- Backpropagation… or more specifically: Gradient Descent using reverse-mode autodiff!
- For each training step:
  - Compute the output error
  - Compute how much each neuron in the previous hidden layer contributed
  - Back-propagate that error in a reverse pass
  - Tweak weights to reduce the error using gradient descent



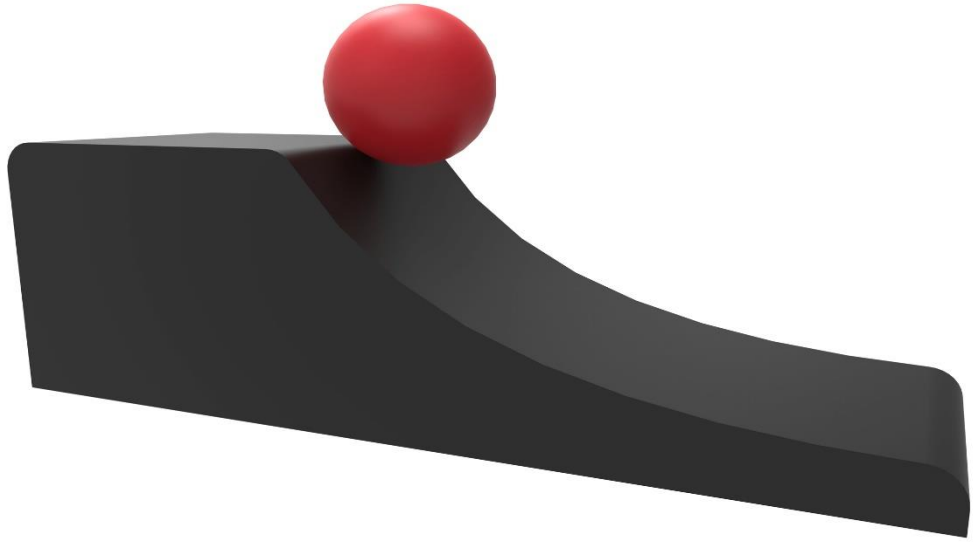
Activation functions (aka rectifier)

- Step functions don't work with gradient descent – there is no gradient!
  - Mathematically, they have no useful derivative.
- Alternatives:
  - Logistic function
  - Hyperbolic tangent function
  - Exponential linear unit (ELU)
  - ReLU function (Rectified Linear Unit)
- ReLU is common. Fast to compute and works well.
  - Also: "Leaky ReLU", "Noisy ReLU"
  - ELU can sometimes lead to faster learning though.



ReLU function

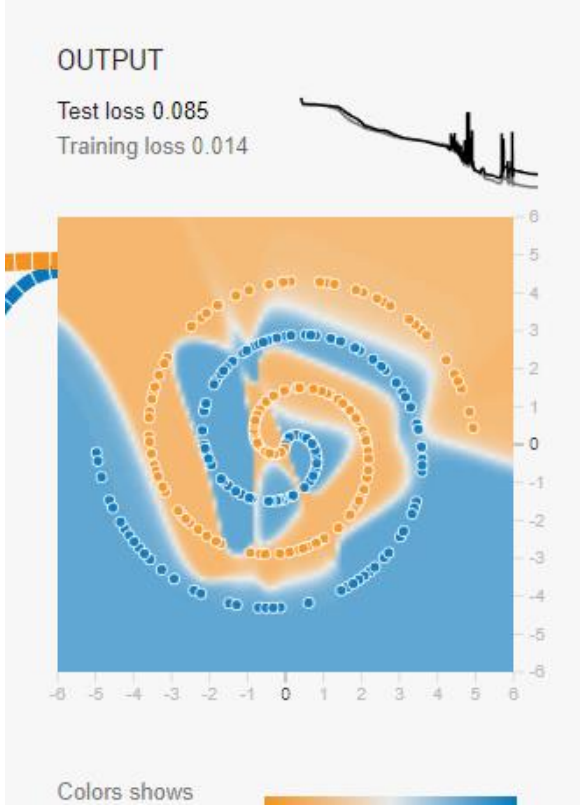
Optimization functions



- There are faster (as in faster learning) optimizers than gradient descent
  - Momentum Optimization
    - Introduces a momentum term to the descent, so it slows down as things start to flatten and speeds up as the slope is steep
  - Nesterov Accelerated Gradient
    - A small tweak on momentum optimization – computes momentum based on the gradient slightly ahead of you, not where you are
  - RMSProp
    - Adaptive learning rate to help point toward the minimum
  - Adam
    - Adaptive moment estimation – momentum + RMSProp combined
    - Popular choice today, easy to use

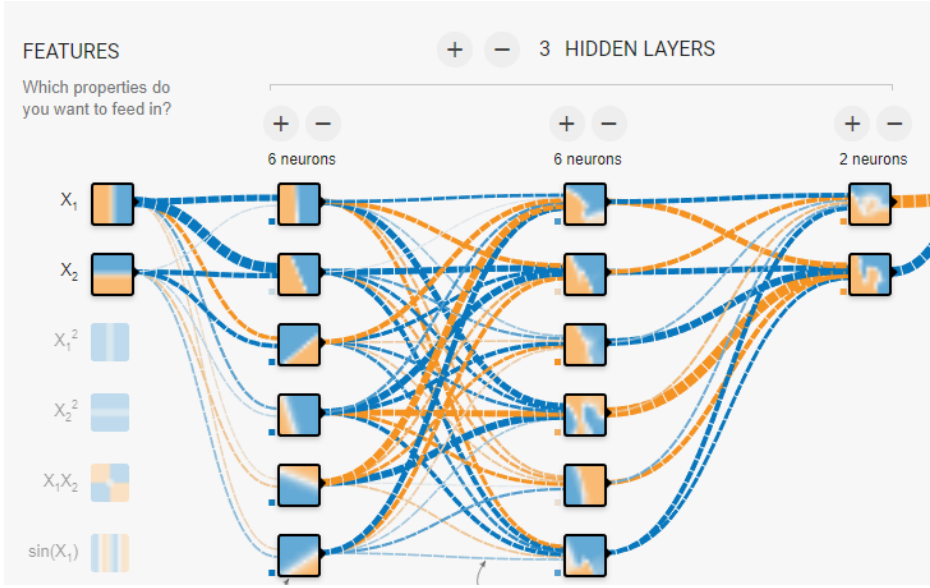
Avoiding Overfitting

- With thousands of weights to tune, overfitting is a problem
- Early stopping (when performance starts dropping)
- Regularization terms added to cost function during training
- Dropout – ignore say 50% of all neurons randomly at each training step
  - Works surprisingly well!
  - Forces your model to spread out its learning



Tuning your topology

- Trial & error is one way
  - Evaluate a smaller network with less neurons in the hidden layers
  - Evaluate a larger network with more layers
    - Try reducing the size of each layer as you progress – form a funnel
- More layers can yield faster learning
- Or just use more layers and neurons than you need, and don't care because you use early stopping.
- Use "model zoos"



Tensorflow

Why Tensorflow?

- It’s not specifically for neural networks– it’s more generally an architecture for executing a graph of numerical operations
- Tensorflow can optimize the processing of that graph, and distribute its processing across a network
  - Sounds a lot like Apache Spark, eh?
- It can also distribute work across GPU’s!
  - Can handle massive scale – it was made by Google
- Runs on about anything
- Highly efficient C++ code with easy to use Python API’s

Tensorflow basics

- Install with pip install tensorflow or pip install tensorflow-gpu
- A tensor is just a fancy name for an array or matrix of values
- To use Tensorflow, you:
  - Construct a graph to compute your tensors
  - Initialize your variables
  - Execute that graph – nothing actually happens until then

World's simplest Tensorflow app:

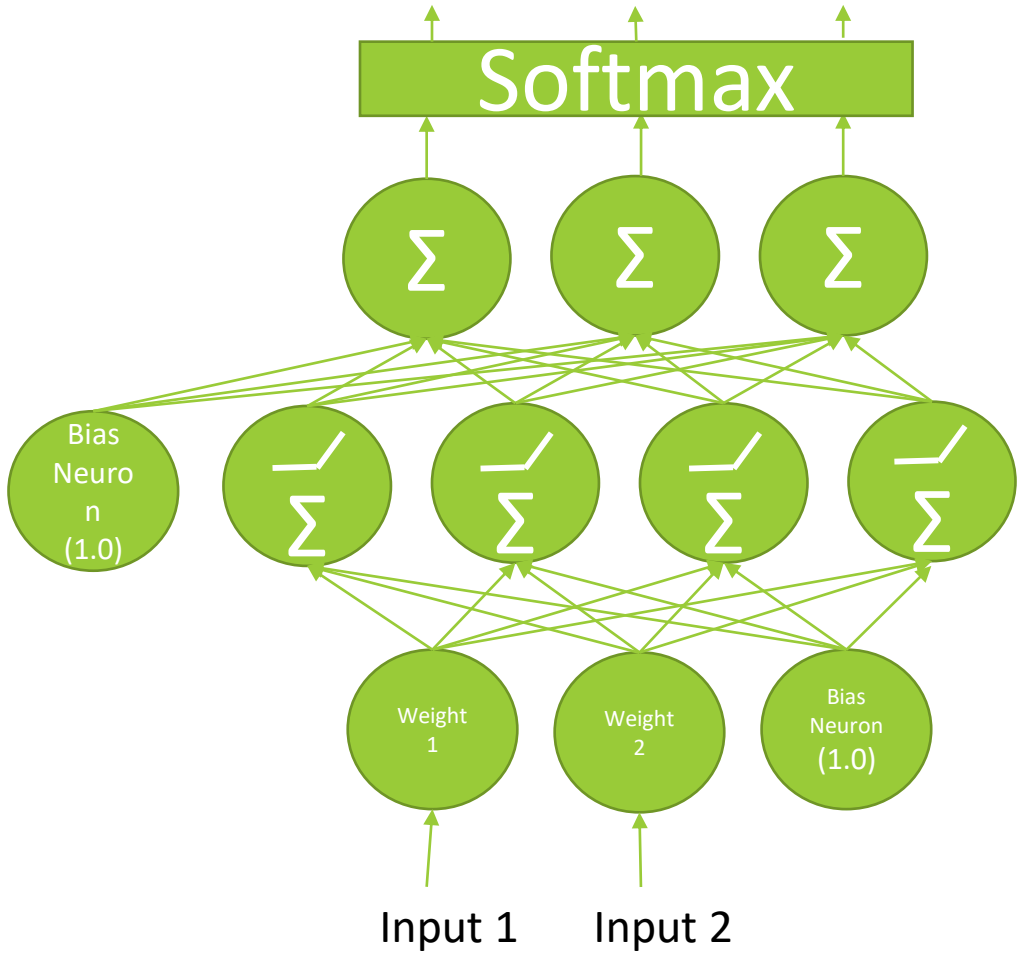
```
import tensorflow as tf

a = tf.Variable(1, name="a")
b = tf.Variable(2, name="b")
f = a + b

init = tf.global_variables_initializer()
with tf.Session() as s:
    init.run()
    print( f.eval() )
```

Creating a neural network with Tensorflow

- Mathematical insights:
  - All those interconnected arrows multiplying weights can be thought of as a big matrix multiplication
  - The bias term can just be added onto the result of that matrix multiplication
- So in Tensorflow, we can define a layer of a neural network as:

```
output = tf.matmul(previous_layer, layer_weights) + layer_biases
```

- By using Tensorflow directly we're kinda doing this the "hard way."



Creating a neural network with Tensorflow

- Load up our training and testing data
- Construct a graph describing our neural network
  - Use **placeholders** for the input data and target labels
    - This way we can use the same graph for training and testing!
  - Use **variables** for the learned weights for each connection and learned biases for each neuron
    - Variables are preserved across runs within a Tensorflow session
- Associate an optimizer (ie gradient descent) to the network
- Run the optimizer with your training data
- Evaluate your trained network with your testing data

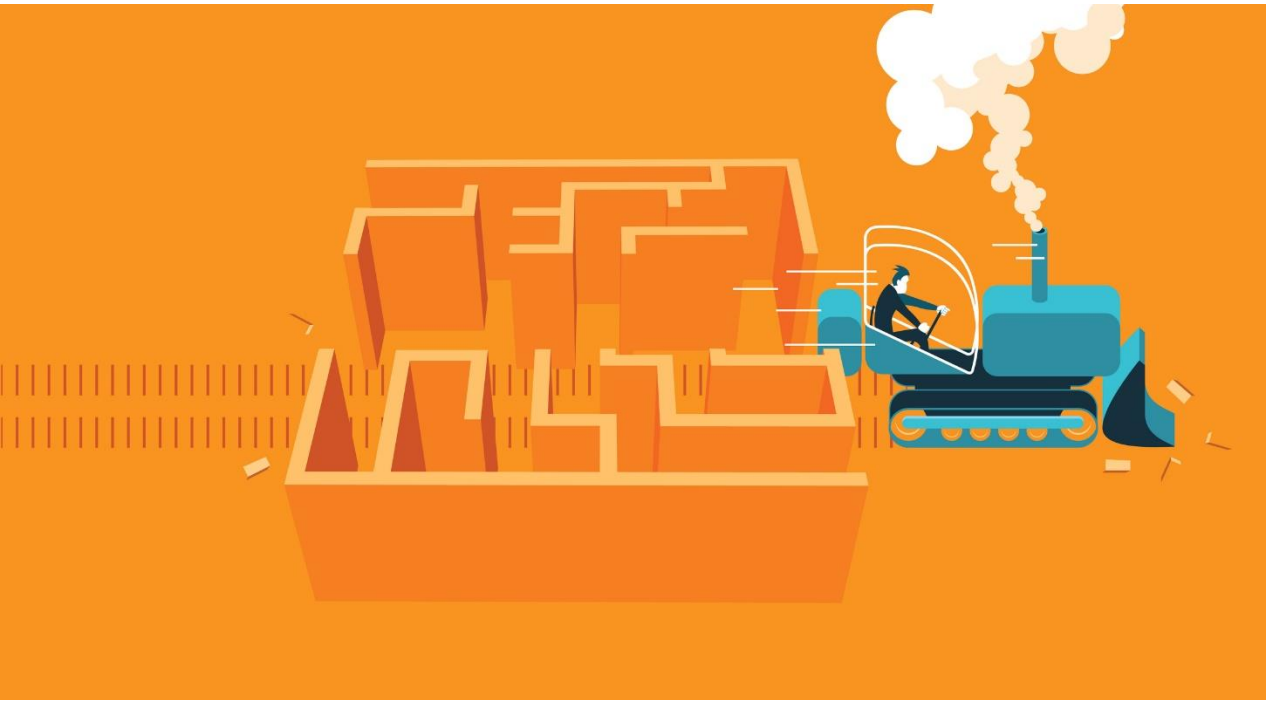
Make sure your features are normalized

- Neural networks usually work best if your input data is normalized.
  - That is, 0 mean and unit variance
  - The real goal is that every input feature is comparable in terms of magnitude
- scikit_learn's StandardScaler can do this for you
- Many data sets are normalized to begin with – such as the one we're about to use.

Let’s try it out

Keras

Why Keras?



- Easy and fast prototyping
  - Available as a higher-level API in Tensorflow 1.9+
  - scikit_learn integration
  - Less to think about – which often yields better results without even trying
  - This is really important! The faster you can experiment, the better your results.

Let’s dive in: MNIST with Keras

Example: multi-class classification

- MNIST is an example of multi-class classification.

```
model = Sequential()

model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9,
        nesterov=True)
model.compile(loss='categorical_crossentropy',
        optimizer=sgd, metrics=['accuracy'])
```

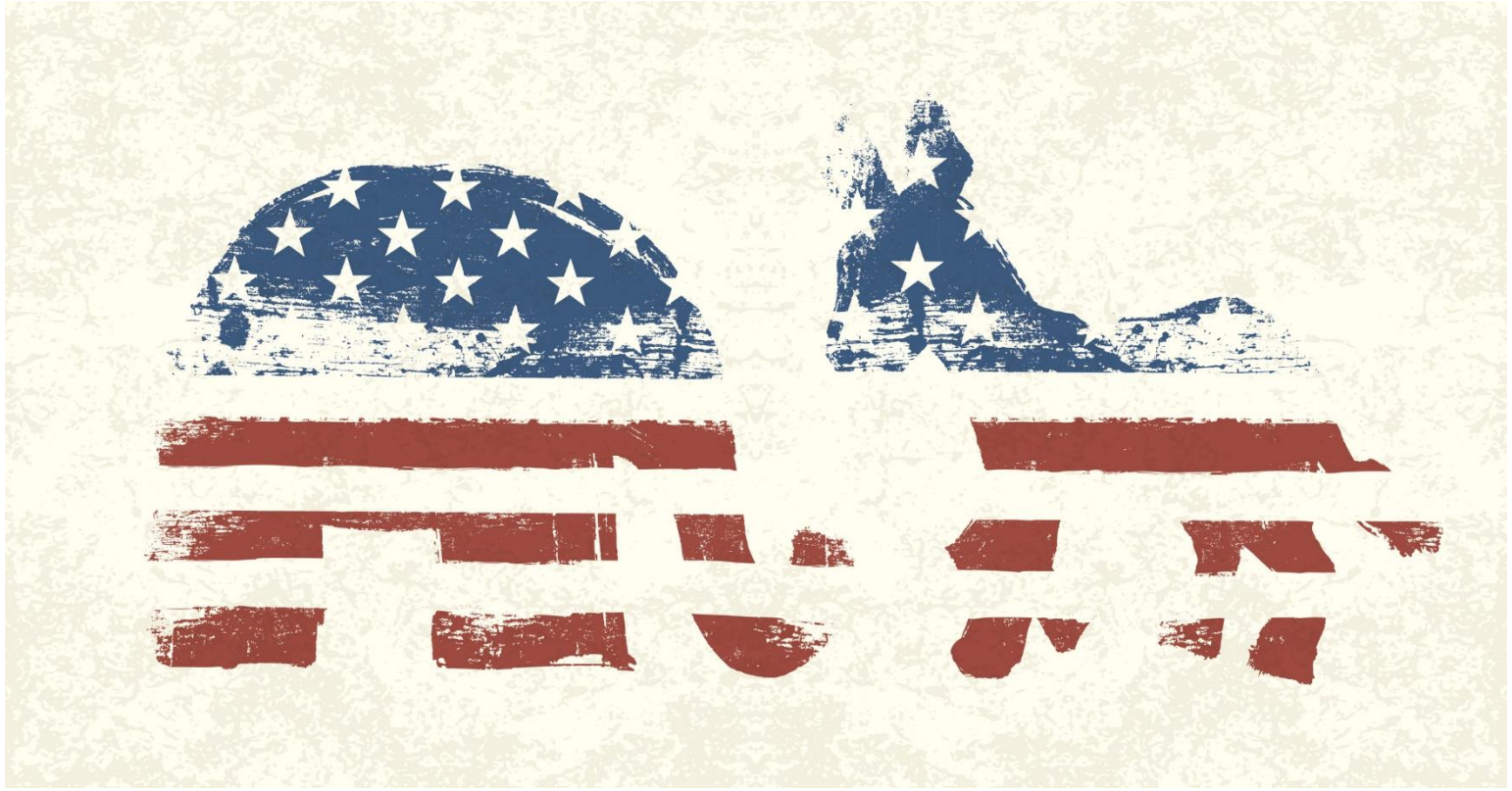
Example: binary classification

```
model = Sequential()
model.add(Dense(64, input_dim=20, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='rmsprop',
        metrics=['accuracy'])
```

Integrating Keras with scikit_learn

```
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

def create_model():
    model = Sequential()
    model.add(Dense(6, input_dim=4, kernel_initializer='normal', activation='relu'))
    model.add(Dense(4, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

estimator = KerasClassifier(build_fn=create_model, epochs, verbose=0)

cv_scores = cross_val_score(estimator, features, labels, cv=10)
print(cv_scores.mean())
```

Let’s try it out: predict political parties with Keras

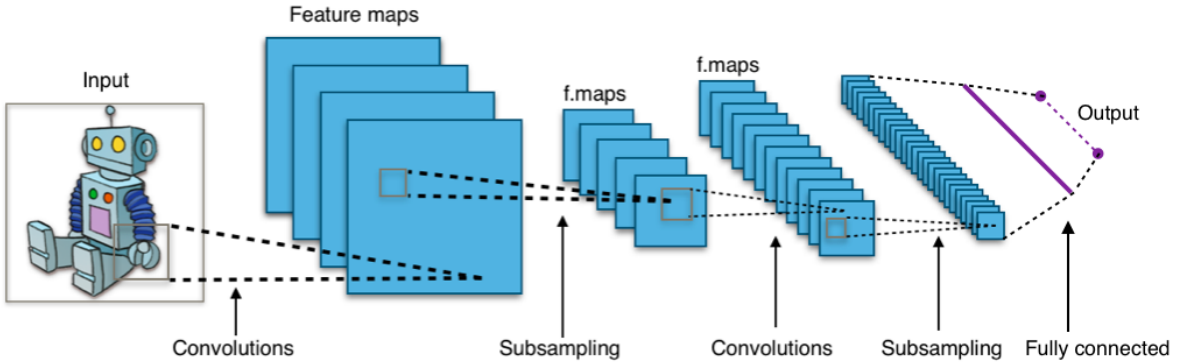


Convolutional Neural Networks

CNN's: what are they for?

- When you have data that doesn't neatly align into columns
  - Images that you want to find features within
  - Machine translation
  - Sentence classification
  - Sentiment analysis
- They can find features that aren't in a specific spot
  - Like a stop sign in a picture
  - Or words within a sentence
- They are "feature-location invariant"

CNN's: how do they work?



- Inspired by the biology of the visual cortex
  - Local receptive fields are groups of neurons that only respond to a part of what your eyes see (subsampling)
  - They overlap each other to cover the entire visual field (convolutions)
  - They feed into higher layers that identify increasingly complex images
    - Some receptive fields identify horizontal lines, lines at different angles, etc. (filters)
    - These would feed into a layer that identifies shapes
    - Which might feed into a layer that identifies objects
  - For color images, extra layers for red, green, and blue

How do we “know” that’s a stop sign?

- Individual local receptive fields scan the image looking for edges, and pick up the edges of the stop sign in a layer
- Those edges in turn get picked up by a higher level convolution that identifies the stop sign’s shape (and letters, too)
- This shape then gets matched against your pattern of what a stop sign looks like, also using the strong red signal coming from your red layers
- That information keeps getting processed upward until your foot hits the brake!
- A CNN works the same way

CNN's with Keras

- Source data must be of appropriate dimensions
  - ie width x length x color channels
- Conv2D layer type does the actual convolution on a 2D image
  - Conv1D and Conv3D also available – doesn't have to be image data
- MaxPooling2D layers can be used to reduce a 2D layer down by taking the maximum value in a given block
- Flatten layers will convert the 2D layer to a 1D layer for passing into a flat hidden layer of neurons
- Typical usage:
  - Conv2D -> MaxPooling2D -> Dropout -> Flatten -> Dense -> Dropout -> Softmax

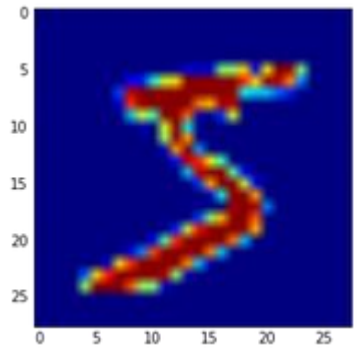
CNN’s are hard

- Very resource-intensive (CPU, GPU, and RAM)
- Lots of hyperparameters
  - Kernel sizes, many layers with different numbers of units, amount of pooling… in addition to the usual stuff like number of layers, choice of optimizer
- Getting the training data is often the hardest part! (As well as storing and accessing it)

Specialized CNN architectures

- Defines specific arrangement of layers, padding, and hyperparameters
- LeNet-5
  - Good for handwriting recognition
- AlexNet
  - Image classification, deeper than LeNet
- GoogLeNet
  - Even deeper, but with better performance
  - Introduces *inception modules* (groups of convolution layers)
- ResNet (Residual Network)
  - Even deeper – maintains performance via *skip connections*.

Let’s try it out

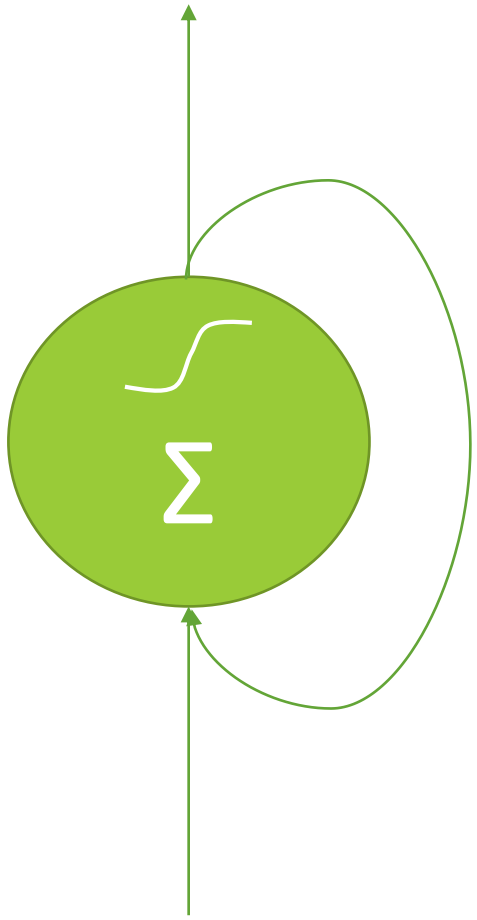


Recurrent Neural Networks

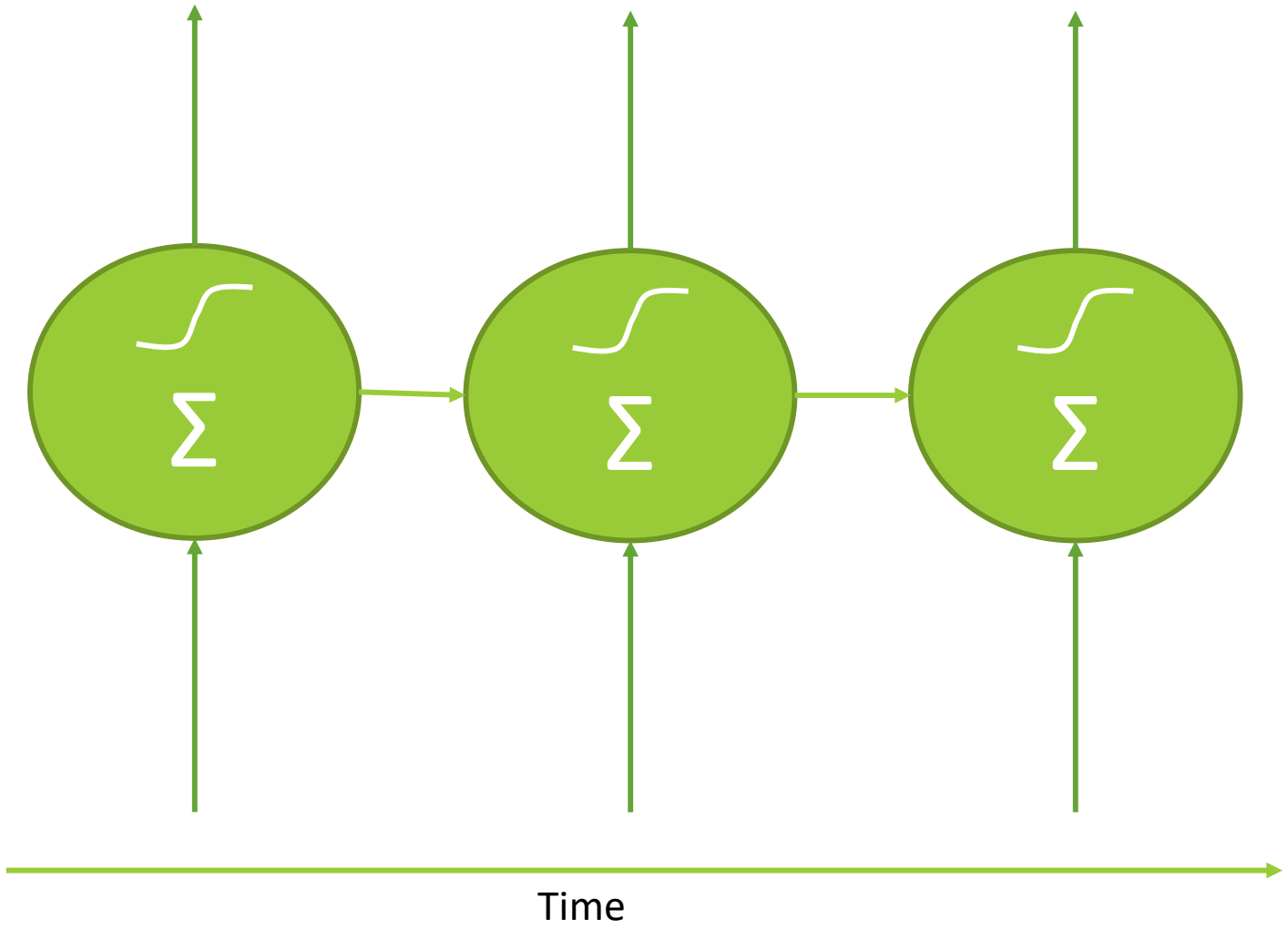
RNN's: what are they for?

- Time-series data
  - When you want to predict future behavior based on past behavior
  - Web logs, sensor logs, stock trades
  - Where to drive your self-driving car based on past trajectories
- Data that consists of sequences of arbitrary length
  - Machine translation
  - Image captions
  - Machine-generated music

A Recurrent Neuron

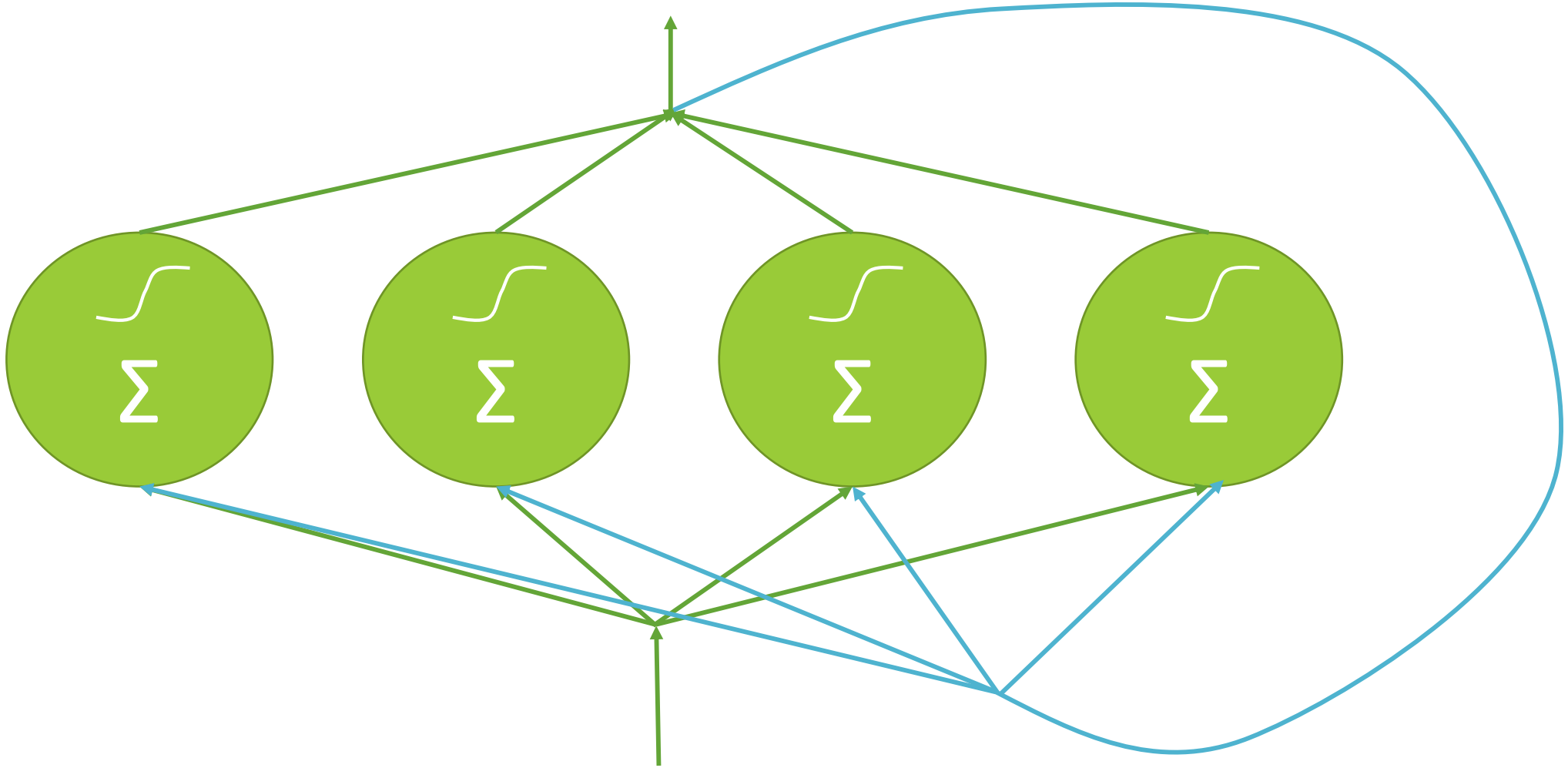


Another Way to Look At It



A “Memory Cell”

A Layer of Recurrent Neurons



RNN Topologies

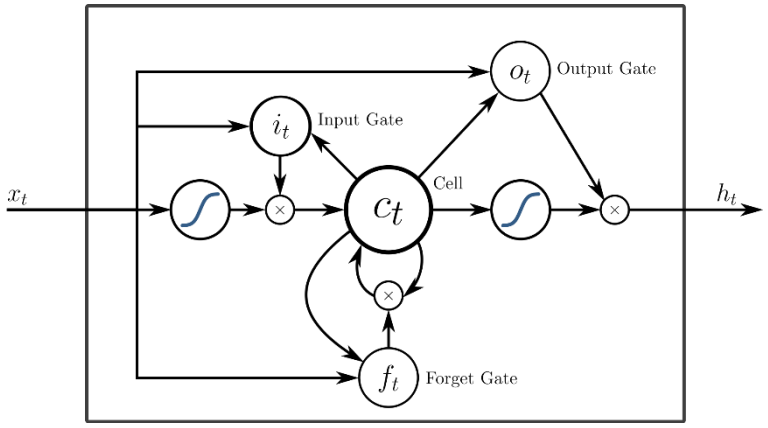
- Sequence to sequence
  - i.e., predict stock prices based on series of historical data
- Sequence to vector
  - i.e., words in a sentence to sentiment
- Vector to sequence
  - i.e., create captions from an image
- Encoder -> Decoder
  - Sequence -> vector -> sequence
  - i.e., machine translation

Training RNN's

- Backpropagation through time
  - Just like backpropagation on MLP's, but applied to each time step.
- All those time steps add up fast
  - Ends up looking like a really, really deep neural network.
  - Can limit backpropagation to a limited number of time steps (truncated backpropagation through time)

Training RNN's

- State from earlier time steps get diluted over time
  - This can be a problem, for example when learning sentence structures
- LSTM Cell
  - Long Short-Term Memory Cell
  - Maintains separate short-term and long-term states
- GRU Cell
  - Gated Recurrent Unit
  - Simplified LSTM Cell that performs about as well



Training RNN's

- It's really hard
  - Very sensitive to topologies, choice of hyperparameters
  - Very resource intensive
  - A wrong choice can lead to a RNN that doesn't converge at all.

Let’s run an example.

The Ethics of Deep Learning

Types of errors

- Accuracy doesn't tell the whole story
- Type 1: False positive
  - Unnecessary surgery
  - Slam on the brakes for no reason
- Type 2: False negative
  - Untreated conditions
  - You crash into the car in front of you

- Think about the ramifications of different types of errors from your model, tune it accordingly

Hidden biases

- Just because your model isn't human doesn't mean it's inherently fair
- Example: train a model on what sort of job applicants get hired, use it to screen resumes
  - Past biases toward gender / age / race will be reflected in your model, because it was reflected in the data you trained the model with.

Is it really better than a human?

- Don’t oversell the capabilities of an algorithm in your excitement
- Example: medical diagnostics that are almost, but not quite, as good as a human doctor
- Another example: self-driving cars that can kill people

Unintended applications of your research

- Gather ‘round the fire while Uncle Frank tells you a story.

Learning More about Deep Learning

Learning more

Final Project

Your Assignment

- Predict if a mass detected in a mammogram is benign or malignant, using the best supervised machine learning model you can find.