Intro to
Machine Learning

John Urbanic

Parallel Computing Scientist
Pittsburgh Supercomputing Center

Distinguished Service Professor
Carnegie Mellon University

Using MLlib

One of the reasons we use spark is for easy access to powerful data analysis tools. The MLlib library gives us a machine learning library that is easy to use and utilizes the scalability of the Spark system.

It has supported APIs for Python (with NumPy), R, Java and Scala.

We will use the Python version in a generic manner that looks very similar to any of the above implementations.

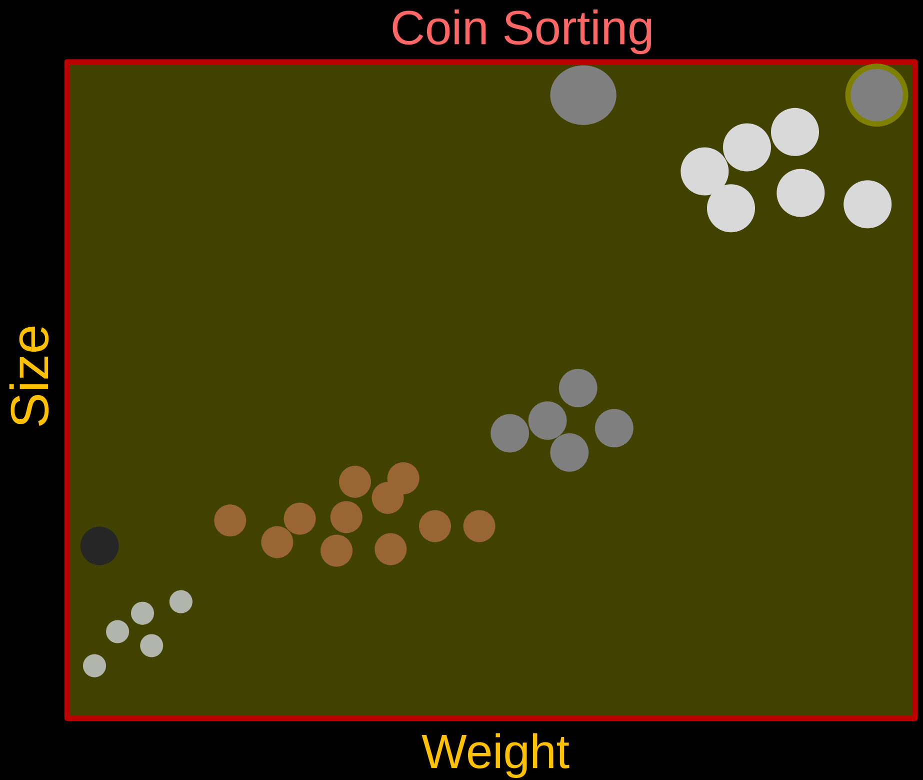
There are good example documents for the clustering routine we are using, as well as alternative clustering algorithms, here:

http://spark.apache.org/docs/latest/mllib-clustering.html

I suggest you use these pages for your Spark work.

Clustering

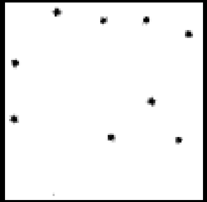
Clustering is a very common operation for finding grouping in data and has countless applications. This is a very simple example, but you will find yourself reaching for a clustering algorithm frequently in pursuing many diverse machine learning objectives, sometimes as one part of a pipeline.



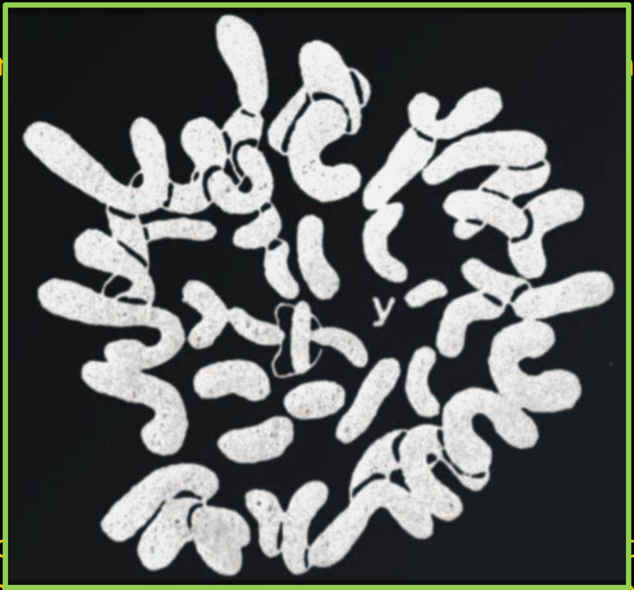
Clustering

As intuitive as clustering is, it presents challenges to implement in an efficient and robust manner.

You might thi ensional spaces.

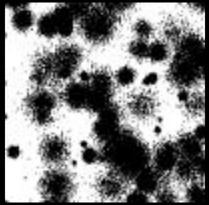


But it can get



From 1900 until 1956 humans were considered to have 48 chromosomes, instead of 46, based upon the interpretation of this camera lucida image.

Sometimes yo tart with. Often you don’t.
How hard can ere?



We will start with 5000 2D points. We want to figure out how many clusters there are, and their centers. Let’s fire up pyspark and get to it…

Finding Clusters

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version
SparkContext availab
>>>
>>> rdd1 = sc.textFi
>>>
>>> rdd2 = rdd1.map(
>>> rdd3 = rdd2.map(
>>>
```

nd integers

```
br06% interact
...
r288%
r288% module load spark
r288% pyspark
```

Make sure you are in the directory with the data file. Otherwise, Spark is dangerously quiet when you textFile() a file that does not exist. It is "lazy" and you won't find out that you have missing data until a later error.

Finding Our Way

```
>>> rdd1 = sc.textFile("5000_points.txt")
>>> rdd1.count()
5000
>>> rdd1.take(4)
['    664159    550946', '    665845    557965', '    597173    575538', '    618600    551446']
>>> rdd2 = rdd1.map(lambda x:x.split())
>>> rdd2.take(4)
[['664159', '550946'], ['665845', '557965'], ['597173', '575538'], ['618600', '551446']]
>>> rdd3 = rdd2.map(lambda x: [int(x[0]),int(x[1])])
>>> rdd3.take(4)
[[664159, 550946], [665845, 557965], [597173, 575538], [618600, 551446]]
>>>
```

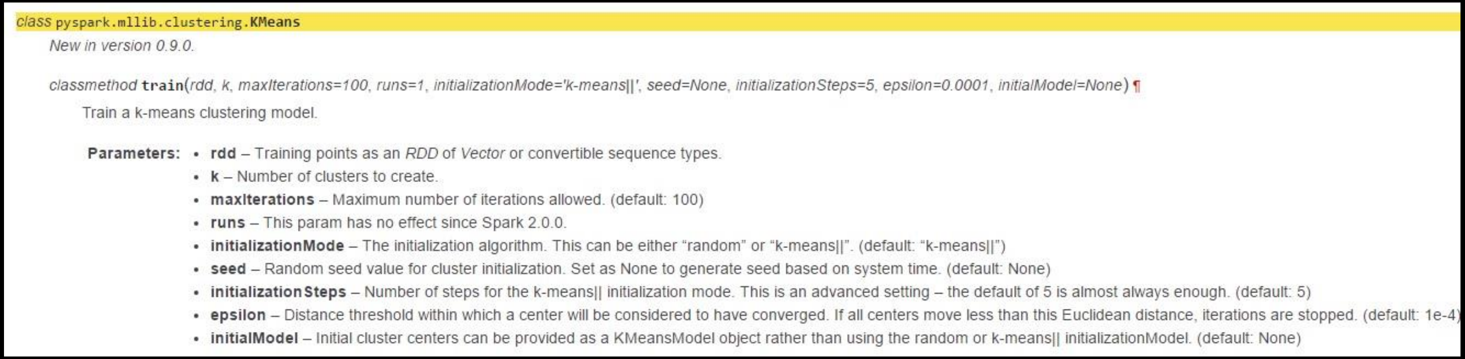
Finding Clusters

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.7.5 (default, Nov 20 2015 02:00:19)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
>>> rdd1 = sc.textFile("5000_points.txt")
>>>
>>> rdd2 = rdd1.map(lambda x:x.split())
>>> rdd3 = rdd2.map(lambda x: [int(x[0]),int(x[1])])
>>>
>>>
>>> from pyspark.mllib.clustering import KMeans
```

Read into RDD

Transform

Import Kmeans



class `pyspark.mllib.clustering.`**`KMeans`**

New in version 0.9.0.

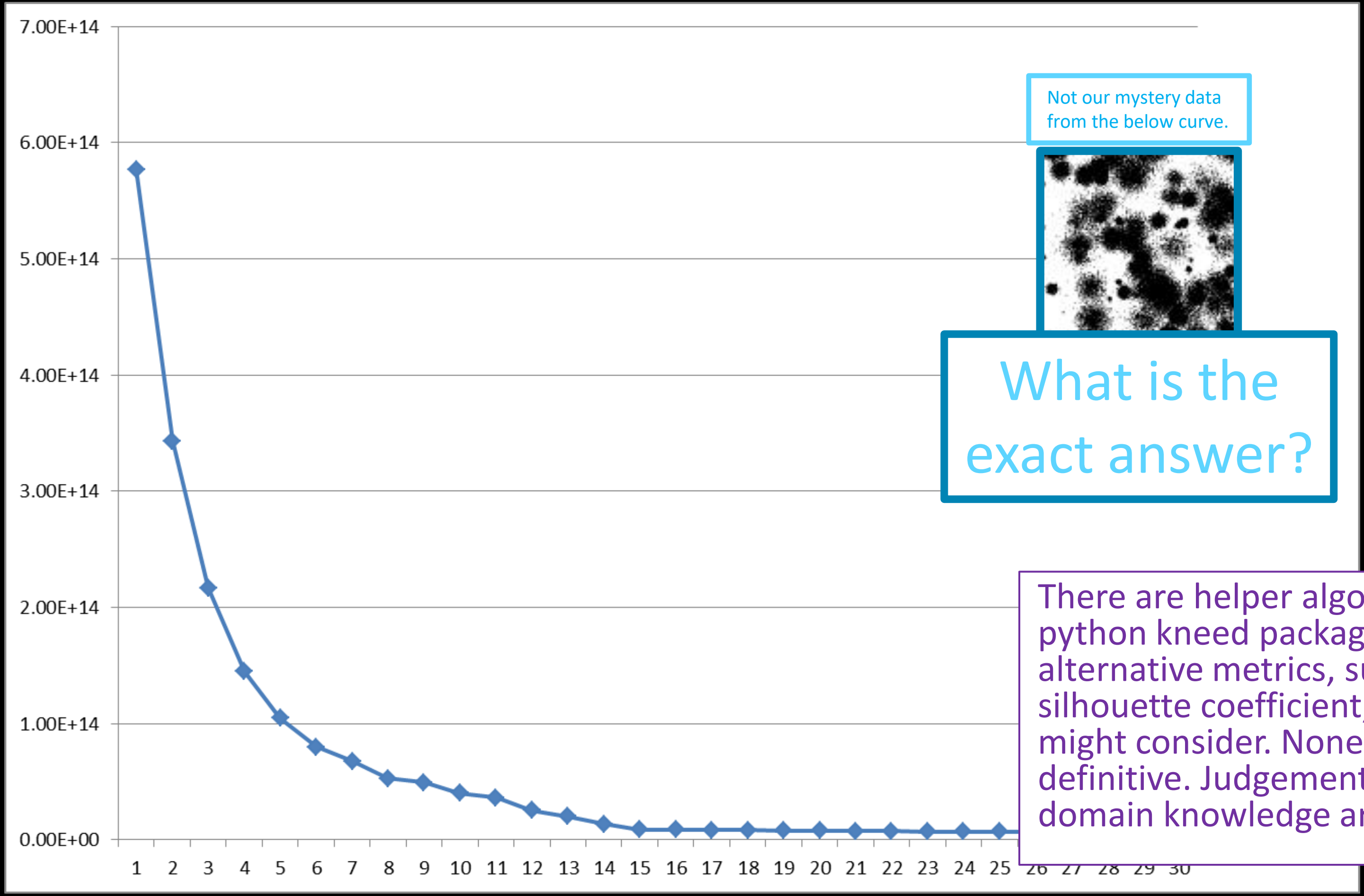
classmethod **`train`**(*rdd*, *k*, *maxIterations=100*, *runs=1*, *initializationMode='k-means||'*, *seed=None*, *initializationSteps=5*, *epsilon=0.0001*, *initialModel=None*) ¶

Train a k-means clustering model.

Parameters:

- **rdd** – Training points as an *RDD* of *Vector* or convertible sequence types.
- **k** – Number of clusters to create.
- **maxIterations** – Maximum number of iterations allowed. (default: 100)
- **runs** – This param has no effect since Spark 2.0.0.
- **initializationMode** – The initialization algorithm. This can be either "random" or "k-means||". (default: "k-means||")
- **seed** – Random seed value for cluster initialization. Set as None to generate seed based on system time. (default: None)
- **initializationSteps** – Number of steps for the k-means|| initialization mode. This is an advanced setting – the default of 5 is almost always enough. (default: 5)
- **epsilon** – Distance threshold within which a center will be considered to have converged. If all centers move less than this Euclidean distance, iterations are stopped. (default: 1e-4)
- **initialModel** – Initial cluster centers can be provided as a KMeansModel object rather than using the random or k-means|| initializationModel. (default: None)

Finding Clusters

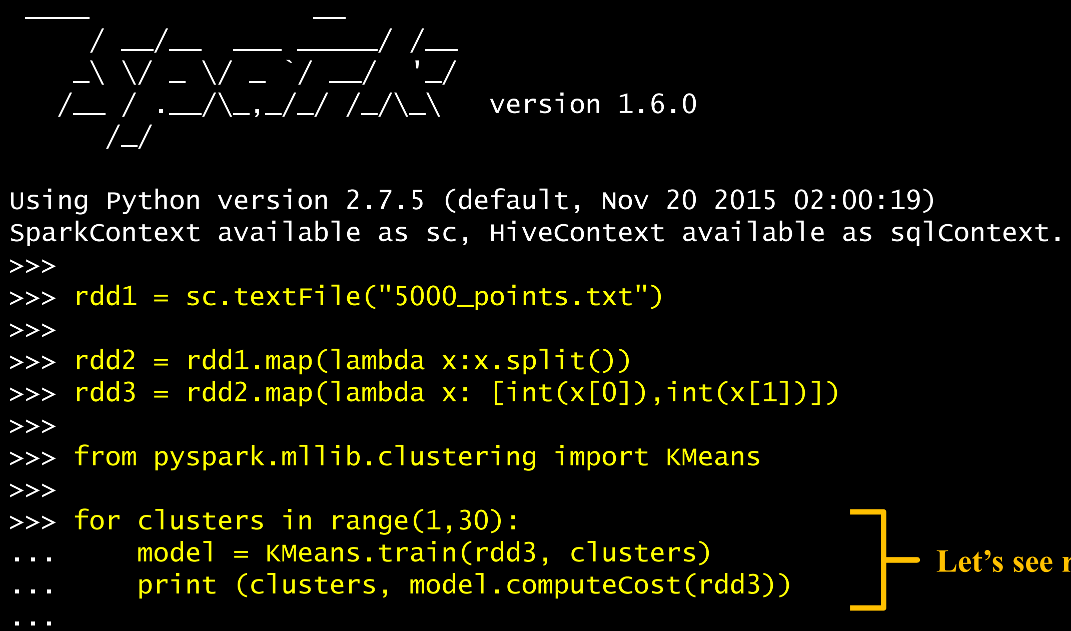


Not our mystery data from the below curve.

What is the exact answer?

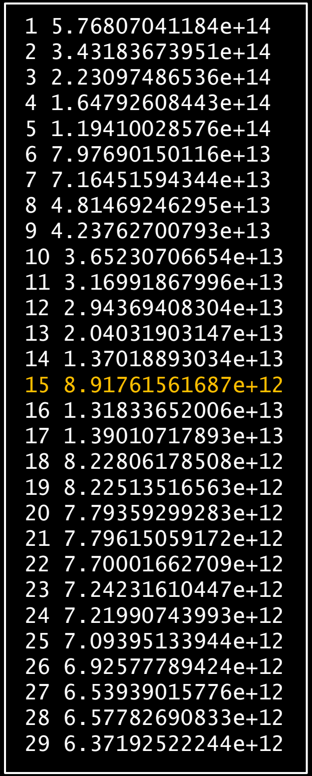
There are helper algorithms (the python kneed package) or alternative metrics, such as the silhouette coefficient, that you might consider. None are definitive. Judgement and domain knowledge are critical.

Finding Clusters



```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.7.5 (default, Nov 20 2015 02:00:19)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
>>> rdd1 = sc.textFile("5000_points.txt")
>>>
>>> rdd2 = rdd1.map(lambda x:x.split())
>>> rdd3 = rdd2.map(lambda x: [int(x[0]),int(x[1])])
>>>
>>> from pyspark.mllib.clustering import KMeans
>>>
>>> for clusters in range(1,30):
...     model = KMeans.train(rdd3, clusters)
...     print (clusters, model.computeCost(rdd3))
...
```

Let’s see results for 1-30 cluster tries



```
1 5.76807041184e+14
2 3.43183673951e+14
3 2.23097486536e+14
4 1.64792608443e+14
5 1.19410028576e+14
6 7.97690150116e+13
7 7.16451594344e+13
8 4.81469246295e+13
9 4.23762700793e+13
10 3.65230706654e+13
11 3.16991867996e+13
12 2.94369408304e+13
13 2.04031903147e+13
14 1.37018893034e+13
15 8.91761561687e+12
16 1.31833652006e+13
17 1.39010717893e+13
18 8.22806178508e+12
19 8.22513516563e+12
20 7.79359299283e+12
21 7.79615059172e+12
22 7.70001662709e+12
23 7.24231610447e+12
24 7.21990743993e+12
25 7.09395133944e+12
26 6.92577789424e+12
27 6.53939015776e+12
28 6.57782690833e+12
29 6.37192522244e+12
```

Right Answer?

```
>>> for trials in range(10):
...     print
...     for clusters in range(12,18):
...         model = KMeans.train(rdd3,clusters)
...         print (clusters, model.computeCost(rdd3))
```

```
12 2.45472346524e+13
13 2.00175423869e+13
14 1.90313863726e+13
15 1.52746006962e+13
16 8.67526114029e+12
17 8.49571894386e+12

12 2.62619056924e+13
13 2.90031673822e+13
14 1.52308079405e+13
15 8.91765957989e+12
16 8.70736515113e+12
17 8.49616440477e+12

12 2.5524719797e+13
13 2.14332949698e+13
14 2.11070395905e+13
15 1.47792736325e+13
16 1.85736955725e+13
17 8.42795740134e+12

12 2.31466242693e+13
13 2.10129797745e+13
14 1.45400177021e+13
15 1.52115329071e+13
16 1.41347332901e+13
17 1.31314086577e+13

12 2.47927778784e+13
13 2.43404436887e+13
14 2.1522702068e+13
15 8.91765000665e+12
16 1.4580927737e+13
17 8.57823507015e+12
```

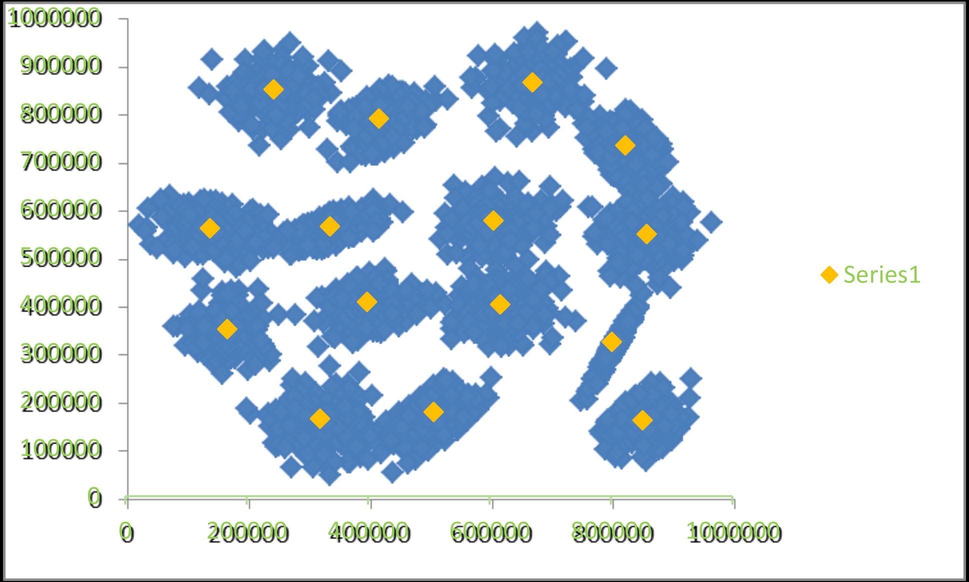
```
12 2.31466520037e+13
13 1.91856542103e+13
14 1.4933202312e+13
15 1.3506302755e+13
16 8.7757678836e+12
17 1.60075548613e+13

12 2.5187054064e+13
13 1.83498739266e+13
14 1.96076943156e+13
15 1.41725666214e+13
16 1.41986217172e+13
17 8.46755159547e+12

12 2.38234539188e+13
13 1.85101922046e+13
14 1.91732620477e+13
15 8.91769396968e+12
16 8.64876051004e+12
17 8.54677681587e+12

12 2.5187054064e+13
13 2.04031903147e+13
14 1.95213876047e+13
15 1.93000628589e+13
16 2.07670831868e+13
17 8.47797102908e+12

12 2.39830397362e+13
13 2.00248378195e+13
14 1.34867337672e+13
15 2.09299321238e+13
16 1.32266735736e+13
17 8.50857884943e+12
```

Find the Centers

```
>>> for trials in range(10):                          #Try ten times to find best result
...    for clusters in range(12, 16):                 #Only look in interesting range
...        model = KMeans.train(rdd3, clusters)
...        cost = model.computeCost(rdd3)
...        centers = model.clusterCenters             #Let’s grab cluster centers
...        if cost<1e+13:                             #If result is good, print it out
...           print (clusters, cost)
...           for coords in centers:
...             print (int(coords[0]), int(coords[1]))
...           break
...
```

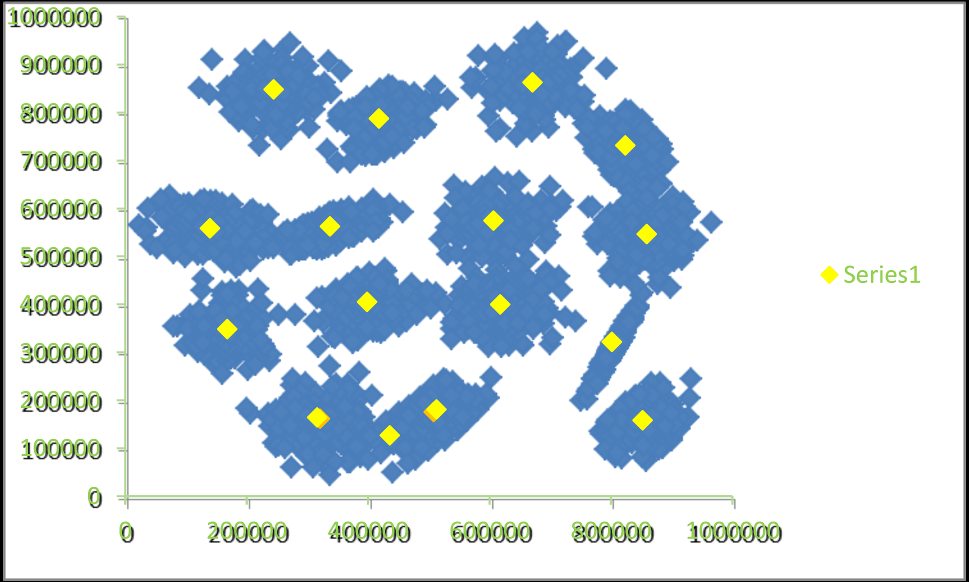
```
15 8.91761561687e+12
852058 157685
606574 574455
320602 161521
139395 558143
858947 546259
337264 562123
244654 847642
398870 404924
670929 862765
823421 731145
507818 175610
801616 321123
617926 399415
417799 787001
167856 347812
15 8.91765957989e+12
670929 862765
139395 558143
244654 847642
852058 157685
617601 399504
801616 321123
507818 175610
337264 562123
858947 546259
823421 731145
606574 574455
167856 347812
398555 404855
417799 787001
320602 161521
```

Fit?



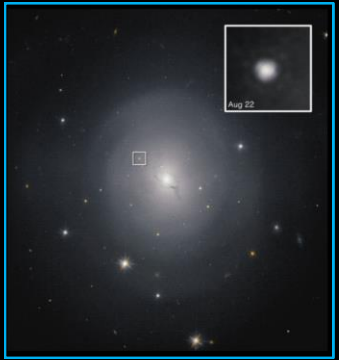
1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
200000
400000
600000
800000
1000000
◆ Series1

16 Clusters


1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
200000
400000
600000
800000
1000000
◆ Series1

We are closer to leading edge science than you might think.

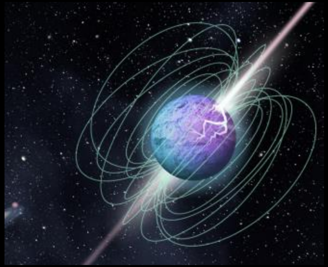
The LIGO gravitational wave detector was able to confirm the collision of two neutron stars with both a gamma ray satellite and optical and other electromagnetic spectrum telescopes. For these transient events, it requires rapid real-time signal analysis to steer other instruments to the proper celestial coordinates. The 2 second gamma-ray burst was detected 1.7 seconds after the GW merger signal. 70 observatories were able to mine signatures in the following days. Even so, the refined location alert took a long time, and much improvement lies ahead.



This rapid processing requirement will only become more extreme as the Square Kilometer Array comes fully on-line. It will generate over an Exabyte of data a day. It will require extreme real-time processing to classify and compress this data down to an archivable size.

Strange, repeating radio signal near the center of the Milky Way has scientists stumped

This article (www.livescience.com/strange-radio-source-milky-way-center) is the summary of the paper (arxiv.org/pdf/2109.00652.pdf) that looks an awful lot like what we are doing.



In April 2020, astronomers picked up some bursts of activity, in the X-ray band of the spectrum, a “run-of-the-mill” magnetar. But the team found that, shortly after the magnetar burst in the X-ray band, CHIME picked up two sharp staccato peaks in the radio band, within several milliseconds of each other, signaling a fast radio burst. The researchers were able to track the radio bursts to a point in the sky that was within a fraction of a degree of SGR 1935+2154 — the same magnetar that was blasting out X-rays around the same time. The team used calibration data from other astrophysical sources to estimate the magnetar’s brightness. They calculated that the magnetar, in the fraction of a second that the FRB flashed, was 3,000 times brighter than any other magnetar radio signal that has yet been observed. Happening in our own galaxy, thousands of times brighter than any other pulse we’ve ever seen.

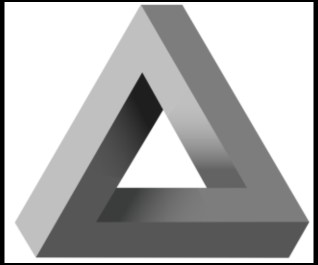
Dimensionality Reduction

We are going to find a recurring theme throughout machine learning:

- Our data naturally resides in higher dimensions
- Reducing the dimensionality makes the problem more tractable
- And simultaneously provides us with insight

This last two bullets highlight the principle that "learning" is often finding an effective compressed representation.

As we return to this theme, we will highlight these slides with our Dimensionality Reduction badge so that you can follow this thread and appreciate how fundamental it is.



Why all these dimensions?

The problems we are going to address, as well as the ones you are likely to encounter, are naturally highly dimensional. If you are new to this concept, lets look at an intuitive example to make it less abstract.

Category	Purchase Total ($)
Children's Clothing	$800
Pet Supplies	$0
Cameras (Dash, Security, Baby)	$450
Containers (Storage)	$350
Romance Book	$0
Remodeling Books	$80
Sporting Goods	$25
Children's Toys	$378
Power Tools	$0
Computers	$0
Garden	$0
Children's Books	$180

< 2900 Categories >

This is a 2900 dimensional vector.

Why all these dimensions?

If we apply our newfound clustering expertise, we might find we have 80 clusters (with an acceptable error).

People spending on “child’s toys “ and “children’s clothing” might cluster with “child’s books” and, less obvious, "cameras (Dashcams, baby monitors and security cams)", because they buy new cars and are safety conscious. We might label this cluster "Young Parents". We also might not feel obligated to label the clusters at all. We can now represent any customer by their distance from these 80 clusters.

80 dimensional vector.

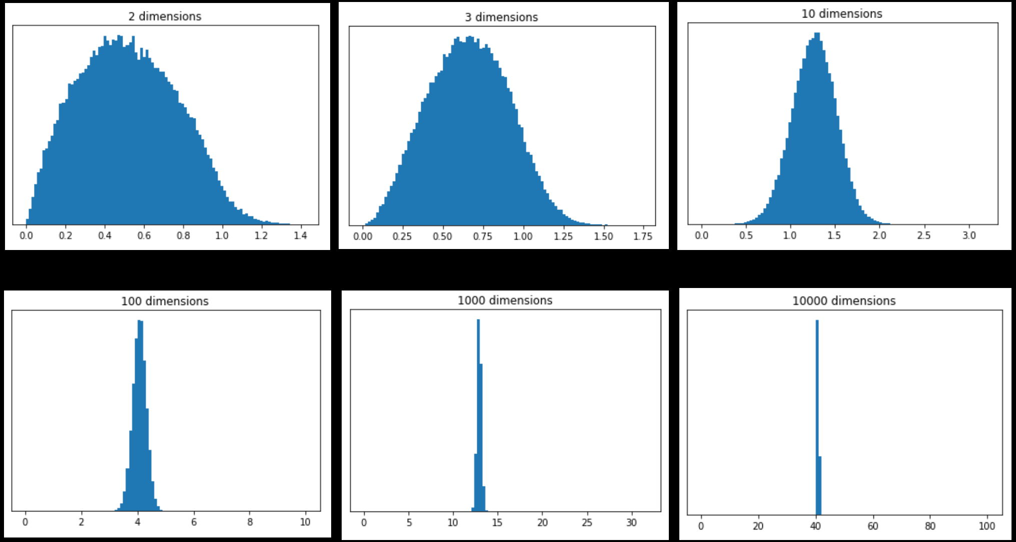
Customer Representation									
Cluster	Young Parents	College Athlete	Auto Enthusiast	Knitter	Steelers Fan	Shakespeare Reader	Sci-Fi Fan	Plumber	…
Distance	0.02	2.3	1.4	8.4	2.2	14.9	3.3	0.8	…

We have now accomplished two things:

- we have compressed our data
- learned something about our customers (who to send a dashcam promo to).

Curse of Dimensionality

This is a good time to point out how our intuition can lead us astray as we increase the dimensionality of our problems - which we will certainly be doing - and to a great degree. There are several related aspects to this phenomenon, often referred to as the *Curse of Dimensionality*. One root cause of confusion is that our notion of Euclidian distance starts to fail in higher dimensions.



These plots show the distributions of pairwise distances between randomly distributed points within differently dimensioned unit hypercubes. Notice how all the points start to be about the same distance apart.

One can imagine this makes life harder on a clustering algorithm!

There are other surprising effects: random vectors are almost all orthogonal; the unit sphere takes almost no volume in the unit square. These cause all kinds of problems when generalizing algorithms from our lowly 3D world.

Most common misuse of this term is to note that points are sparser in a high number of features. This is true, but not this effect.

Metrics

Even the definition of distance (the *metric*) can vary based upon application. If you are solving chess problems, you might find the Manhattan distance (or taxicab metric) to be most useful.

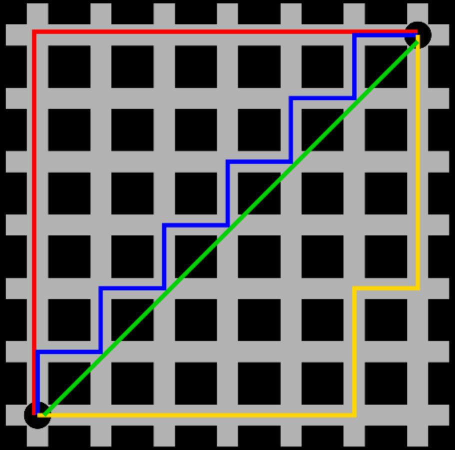


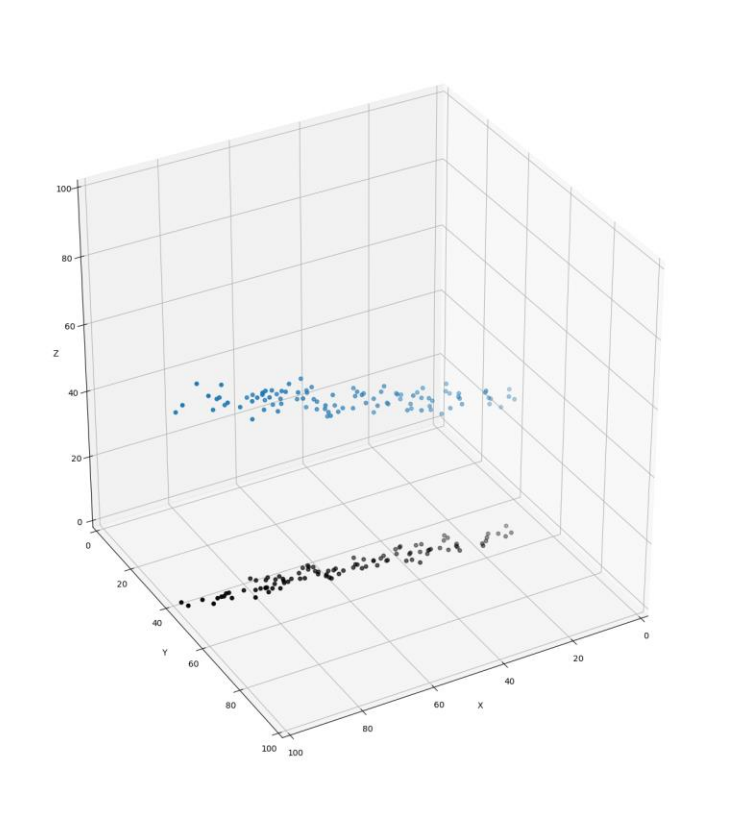
Image Source: Wikipedia

For comparing text strings, we might choose one of dozens of different metrics. For spell checking you might want one that is good for phonetic distance, or maybe edit distance. For natural language processing (NLP), you probably care more about tokens.

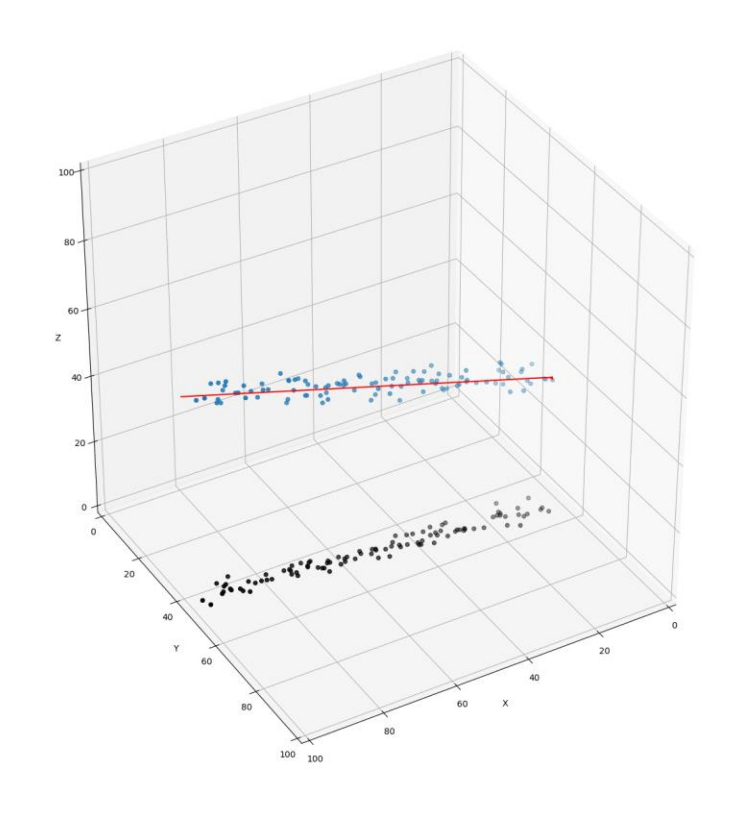
For genomics, you might care more about string sequences.

Some useful measures don't even qualify as metrics (usually because they fail the triangle inequality: $a + b \geq c$).

Everybody's Favorite DR: Principal Component Analysis

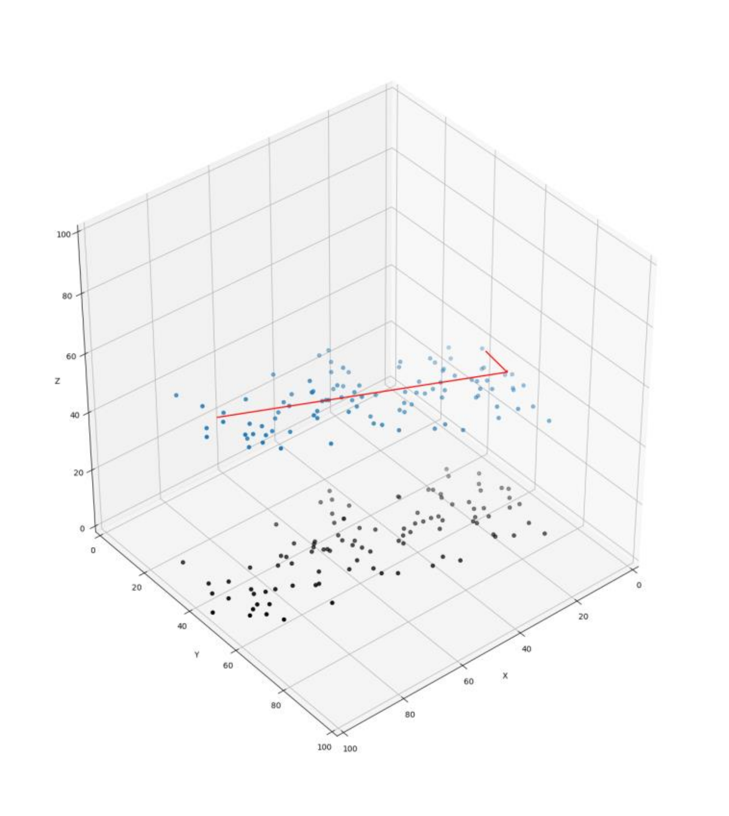


3D Data Set

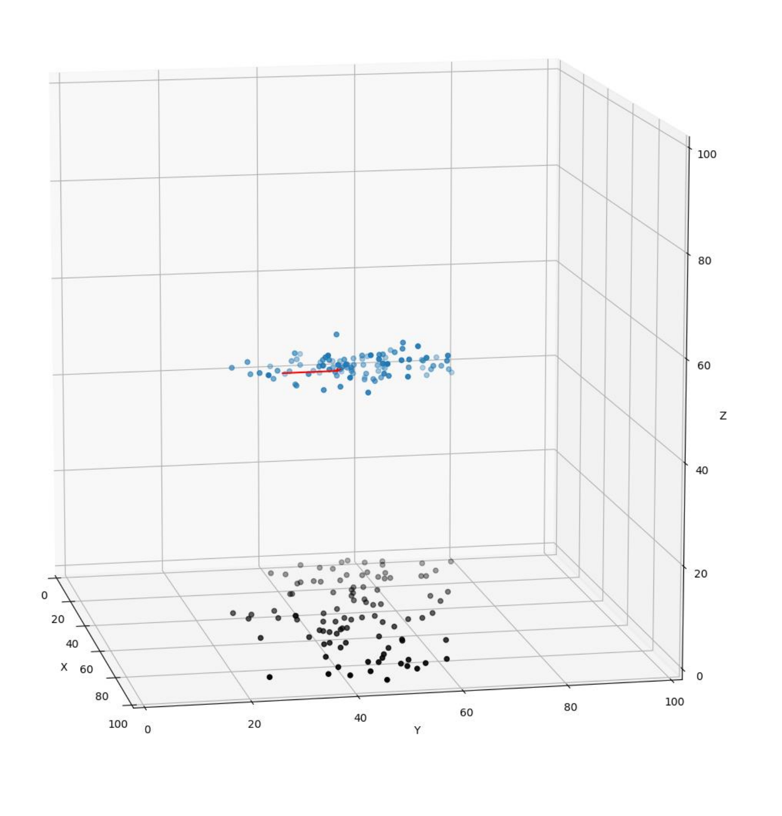


Maybe mostly 1D!

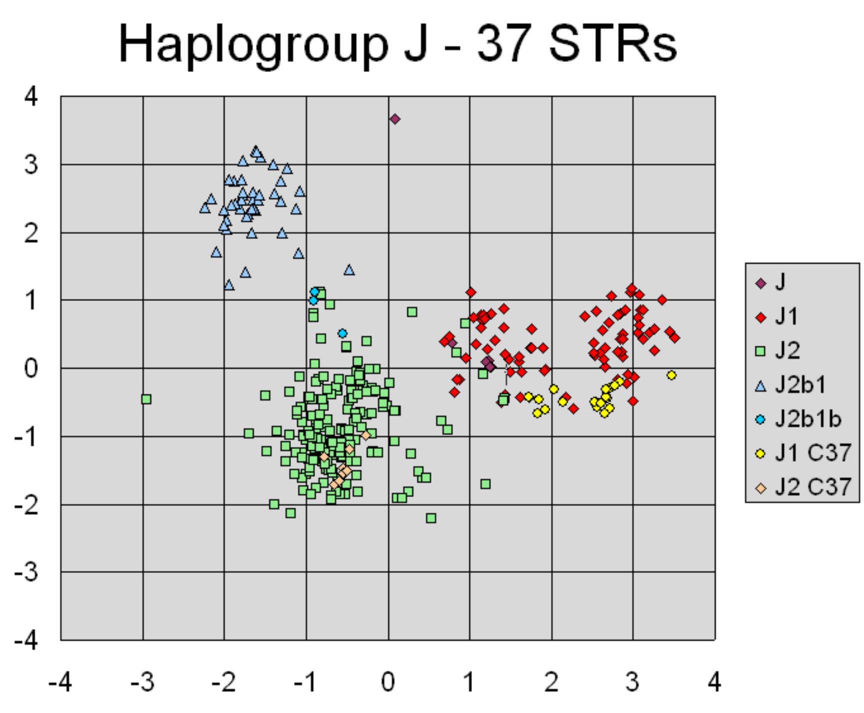
Alternative DR: Principal Component Analysis



Flatter 2D-ish Data Set



View down the 1st Princ. Comp.



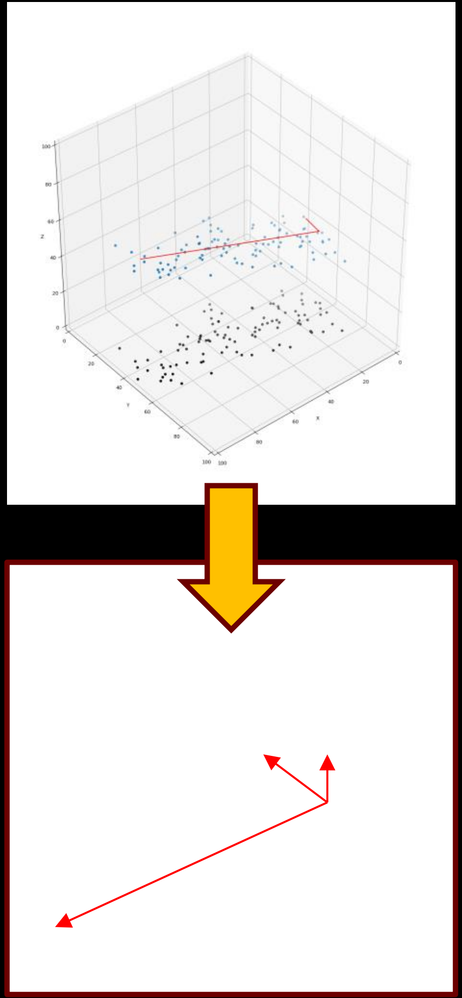
Principal Component Analysis plot for 354 Y-chromosome haplotypes from the public Ysearch database, identified as belonging to Y-DNA Haplogroup J and its subgroups. Data represent first and second most significant components calculated from 37 Y-STR markers. The different J1, J2 and J2b subhaplogroups are resolved clearly into different clusters. Additional structure can be revealed by calculating further PCAs for each of the subgroup memberships separately. The labels J1 C37 and J2 C37 identify haplotypes often associated with Cohen lineages.View down the 1st Princ. Comp. *This is a Wikipedia example of a PCA use case.*

Principal Component *Analysis*

Very often, PCA users get fixated on the *Analysis* part of PCA, and get confused at to the very basic information that PCA actually extracts from the data, the Principal Components.

While we are often inputting large and complex data-sets, and the visible end result is usually a 2D graph (because we all like a nice 2D graph), this is really just one application of the more fundamental principle components.

Principal Components

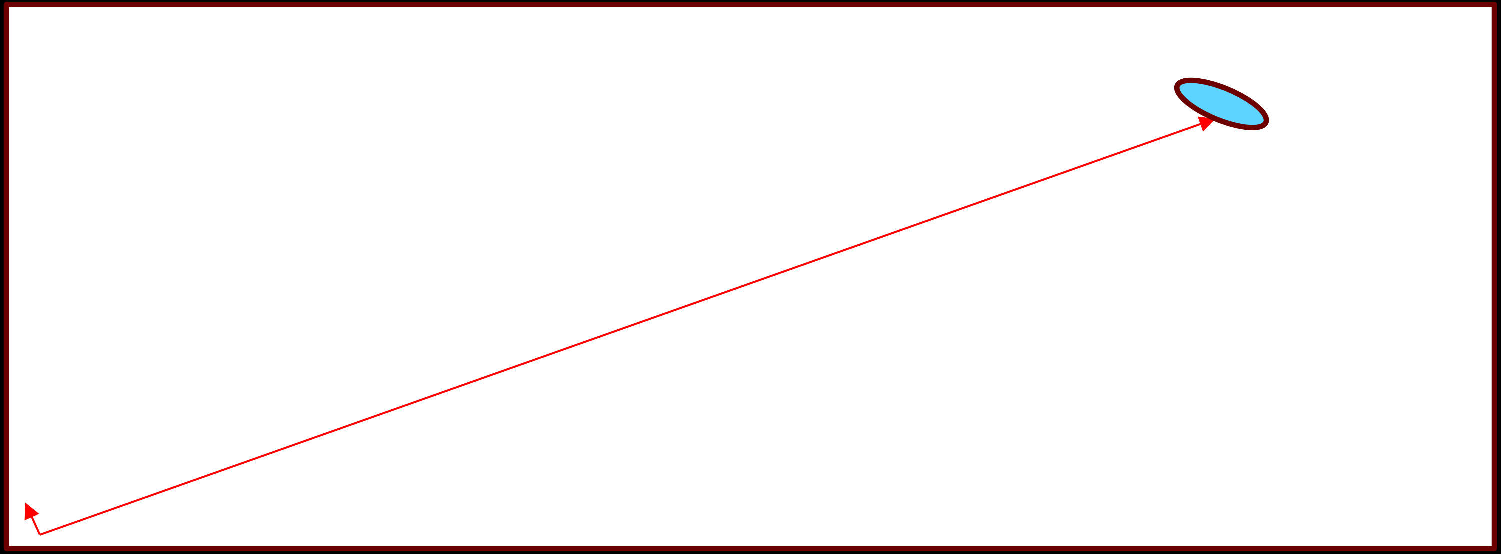


But, don't let yourself get distracted by all the dots! The knowledge of the algorithm is completely described by a set of vectors of the same dimensionality as the input data. If you have 6D data, you get back 6 6D vectors. The data is 3D in the case at left.

Those vectors have a magnitude proportional to the explained variance. That is useful guidance as to how much you can reduce the dimensionality.

If you tell the algorithm to project the data down to some lower dimension (2D or 3D typically, if not always meaningfully), those vectors give us the linear transform we use to do so.

What can't the components tell you?

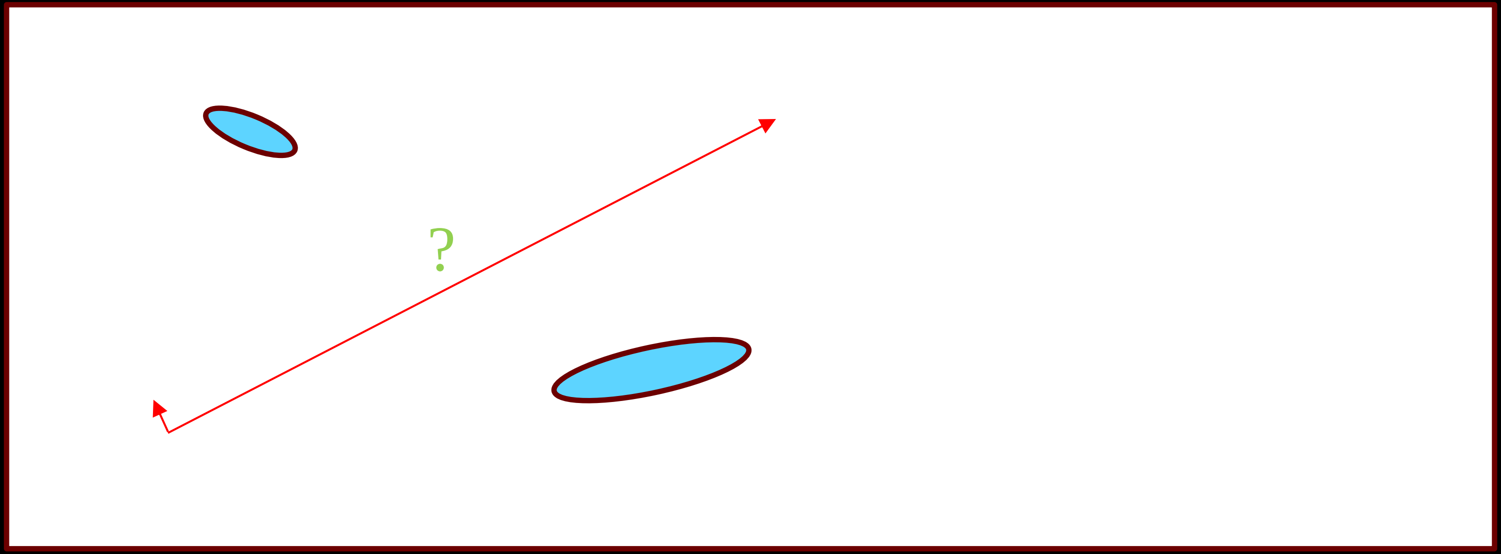


If you have very "uncentered" data, your first principle component is likely to correspond to the mean of the data. So, you probably want to center your data first.

Similarly, if you have very different scales for your axis, it will be difficult to get meaningful results. So, you may want to rescale. However, be aware that your results are now scaled!

And, if your axis have different units (say temperature and mass), you need to be very thoughtful. Switching from Fahrenheit to Celsius would give different results!

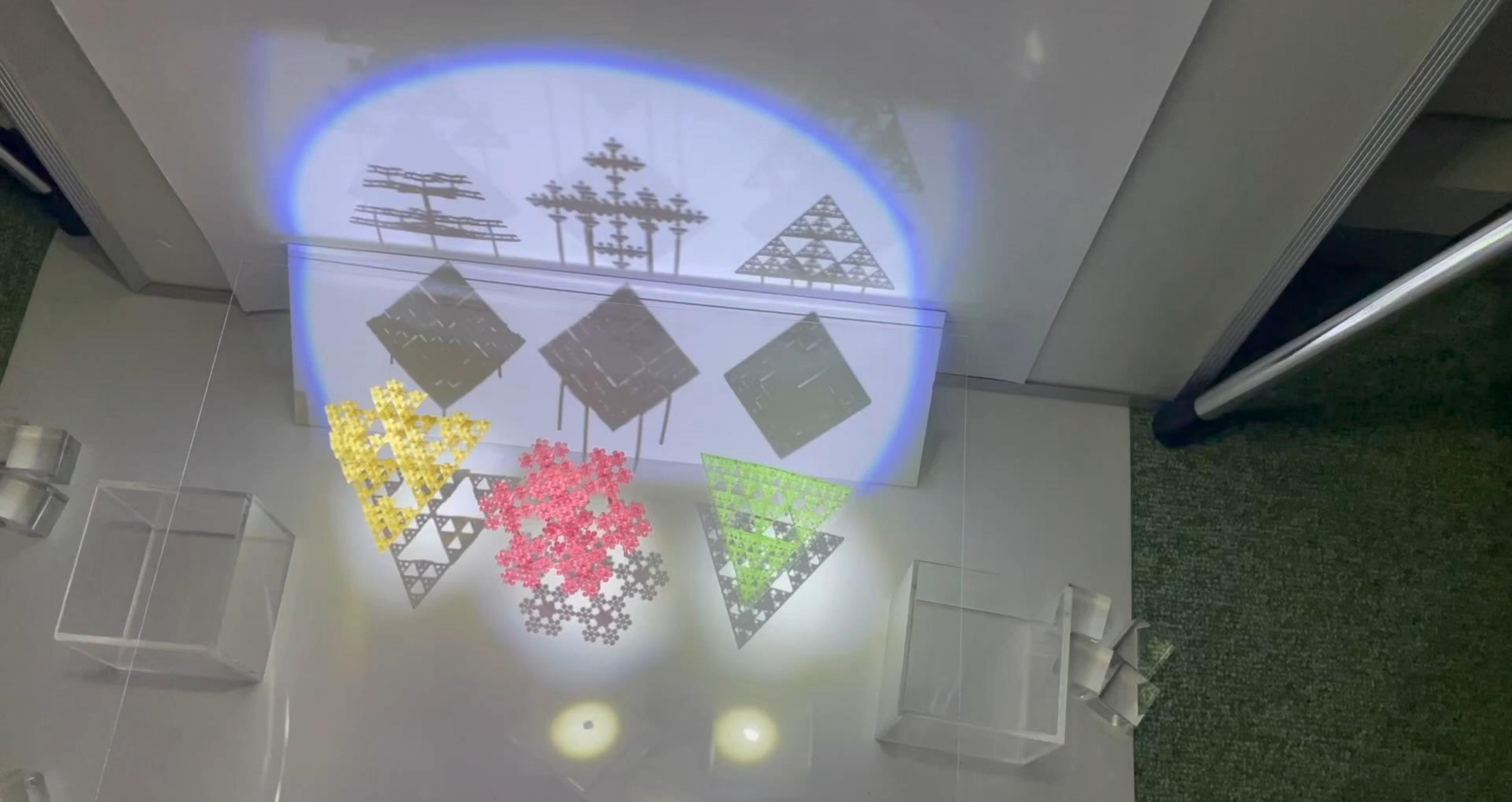
What can't the components tell you?



And ask yourself what you expect PCA to do with it's limited output of vectors. If you have a split, or very non-linear distribution, how do you expect it to respond? It there isn't a good answer, then you need to reconsider your approach. Or at least have some suspicion of your results.

Everything casts shadows! Those shadows can be very deceiving.

From the fun channel of Hideki Tsuiki



Shadow Showtime!

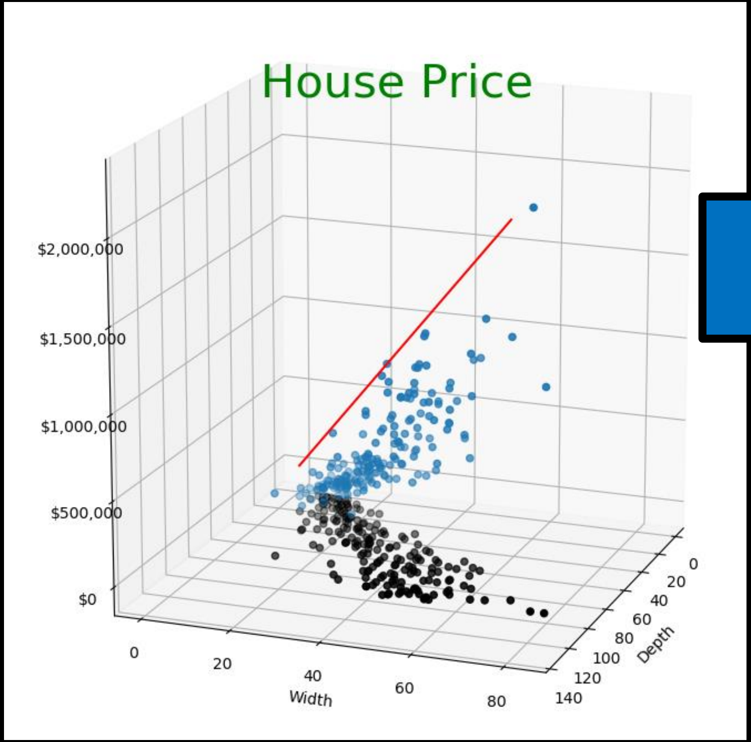
Why So Many Alternatives?

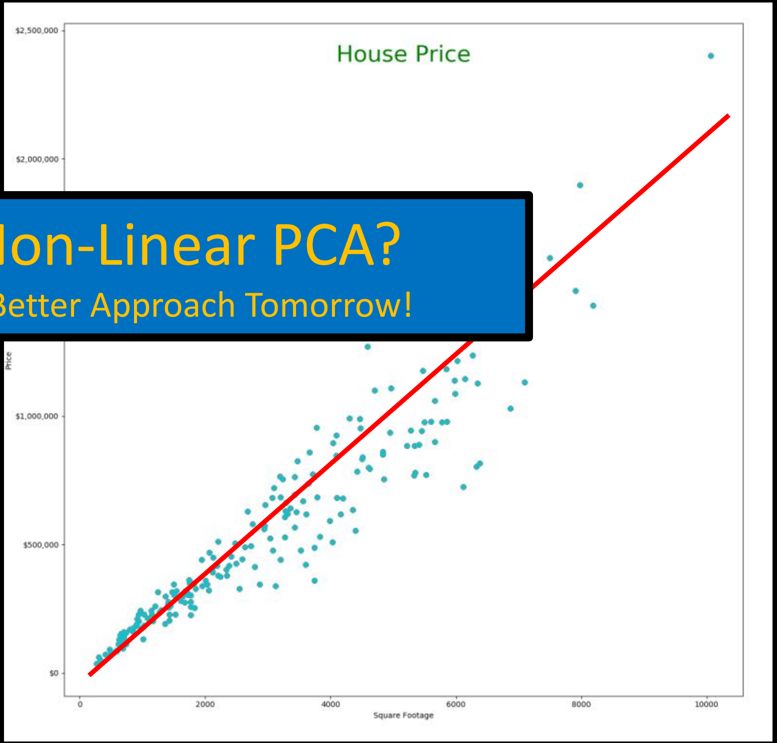
Let's look at one more example today. Suppose we are tying to do a Zillow type of analysis and predict home values based upon available factors. We may have an entry (vector) for each home that captures this kind of data:

Home Data	
Latitude	4833438 north
Longitude	630084 east
Last Sale Price	$ 480,000
Last Sale Year	1998
Width	62
Depth	40
Floors	3
Bedrooms	3
Bathrooms	2
Garage	2
Yard Width	84
Yard Depth	60
...	...

There may be some opportunities to reduce the dimension of the vector here. Perhaps clustering on the geographical coordinates...

Principal Component Analysis Fail





Non-Linear PCA?

A Better Approach Tomorrow!

1st Component Off

Data Not Very Linear

D x W Is Not Linear

But (DxW) Fits Well

Why the fascination with linear techniques?

The Streetlight Effect

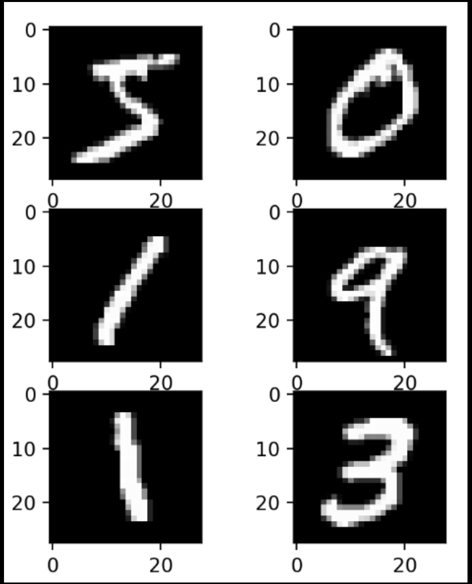
This is a very real and powerful force throughout the sciences.

It is not because practitioners are dumb.

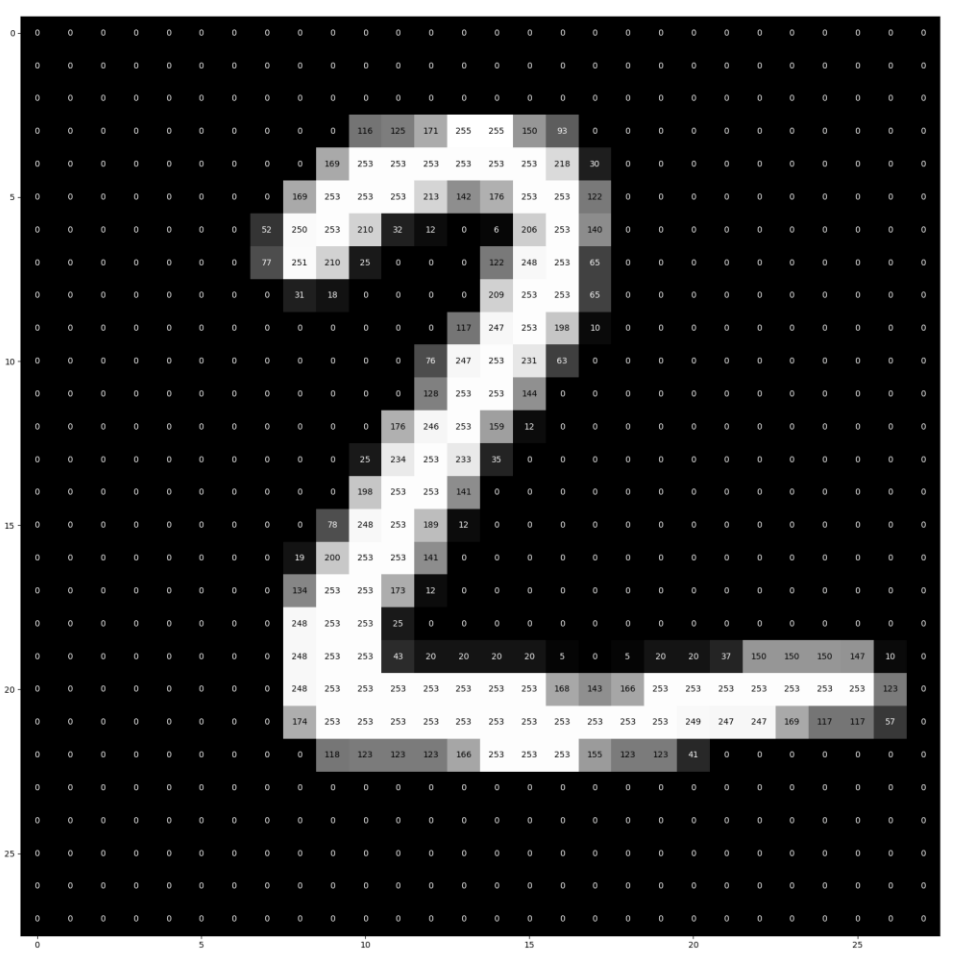
But, it is also very often neither explained nor justified.

Which leads to great confusion.

Why Would An Image Have 784 Dimensions?

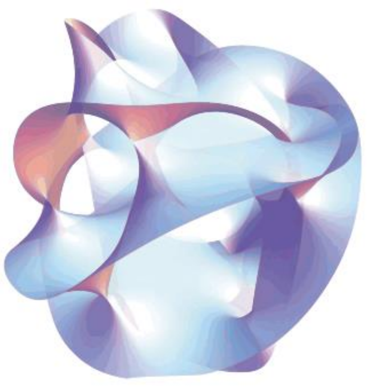


MNIST 28x28
greyscale images

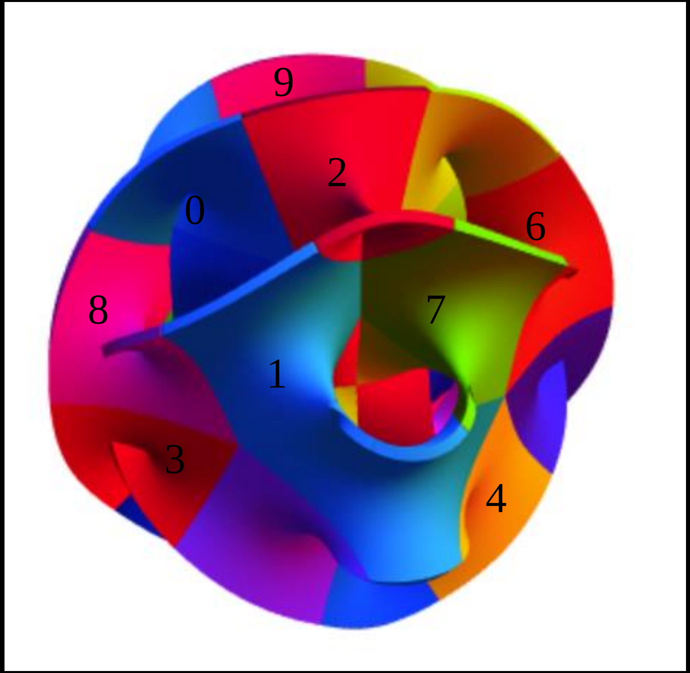

0
5
10
15
20
25
0
5
10
15
20
25

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	116	125	171	255	255	150	93	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	253	253	253	253	253	218	30	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	169	253	253	253	213	142	176	253	253	122	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	52	250	253	210	32	12	0	6	206	253	140	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	77	251	210	25	0	0	0	122	248	253	65	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	31	18	0	0	0	0	209	253	253	65	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	117	247	253	198	10	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	76	247	253	231	63	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	128	253	253	144	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	176	246	253	159	12	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	25	234	253	233	35	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	198	253	253	141	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	78	248	253	189	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	19	200	253	253	141	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	134	253	253	173	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	248	253	253	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	248	253	253	43	20	20	20	20	5	0	5	20	20	37	150	150	150	147	10	0
0	0	0	0	0	0	0	0	248	253	253	253	253	253	253	253	168	143	166	253	253	253	253	253	253	253	123	0
0	0	0	0	0	0	0	0	174	253	253	253	253	253	253	253	253	253	253	253	249	247	247	169	117	117	57	0
0	0	0	0	0	0	0	0	0	118	123	123	123	166	253	253	253	155	123	123	41	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

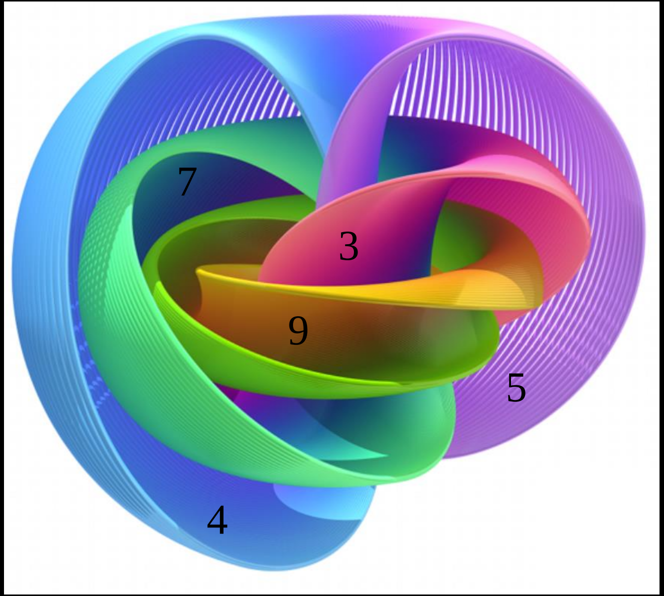
Central Hypothesis of Modern DL



Data Lives On
A Lower Dimensional
Manifold



Maybe Very Contiguous

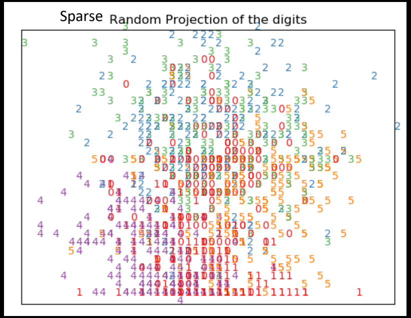


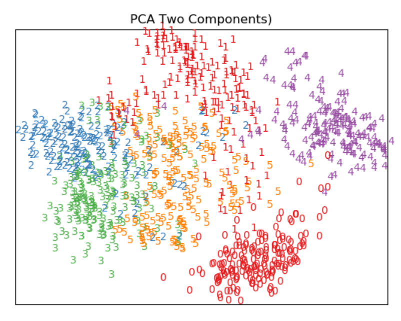
Maybe A Small Set
Of Disconnected

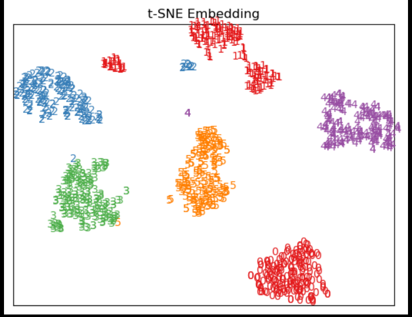
Images from Wikipedia

Testing These Ideas With Scikit-learn

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import (datasets, decomposition, manifold, random_projection)

def draw(X, title):
    plt.figure()
    plt.xlim(X.min(0)[0],X.max(0)[0]); plt.ylim(X.min(0)[1],X.max(0)[1])
    plt.xticks([]); plt.yticks([])
    plt.title(title)
    for i in range(X.shape[0]):
        plt.text(X[i, 0], X[i, 1], str(y[i]), color=plt.cm.Set1(y[i] / 10.) )

digits = datasets.load_digits(n_class=6)
X = digits.data
y = digits.target

rp = random_projection.SparseRandomProjection(n_components=2, random_state=42)
X_projected = rp.fit_transform(X)
draw(X_projected, "Sparse Random Projection of the digits")

X_pca = decomposition.PCA(n_components=2).fit_transform(X)
draw(X_pca, "PCA (Two Components)")

tsne = manifold.TSNE(n_components=2, init='pca', random_state=0)
X_tsne = tsne.fit_transform(X)
draw(X_tsne, "t-SNE Embedding")

plt.show()
```

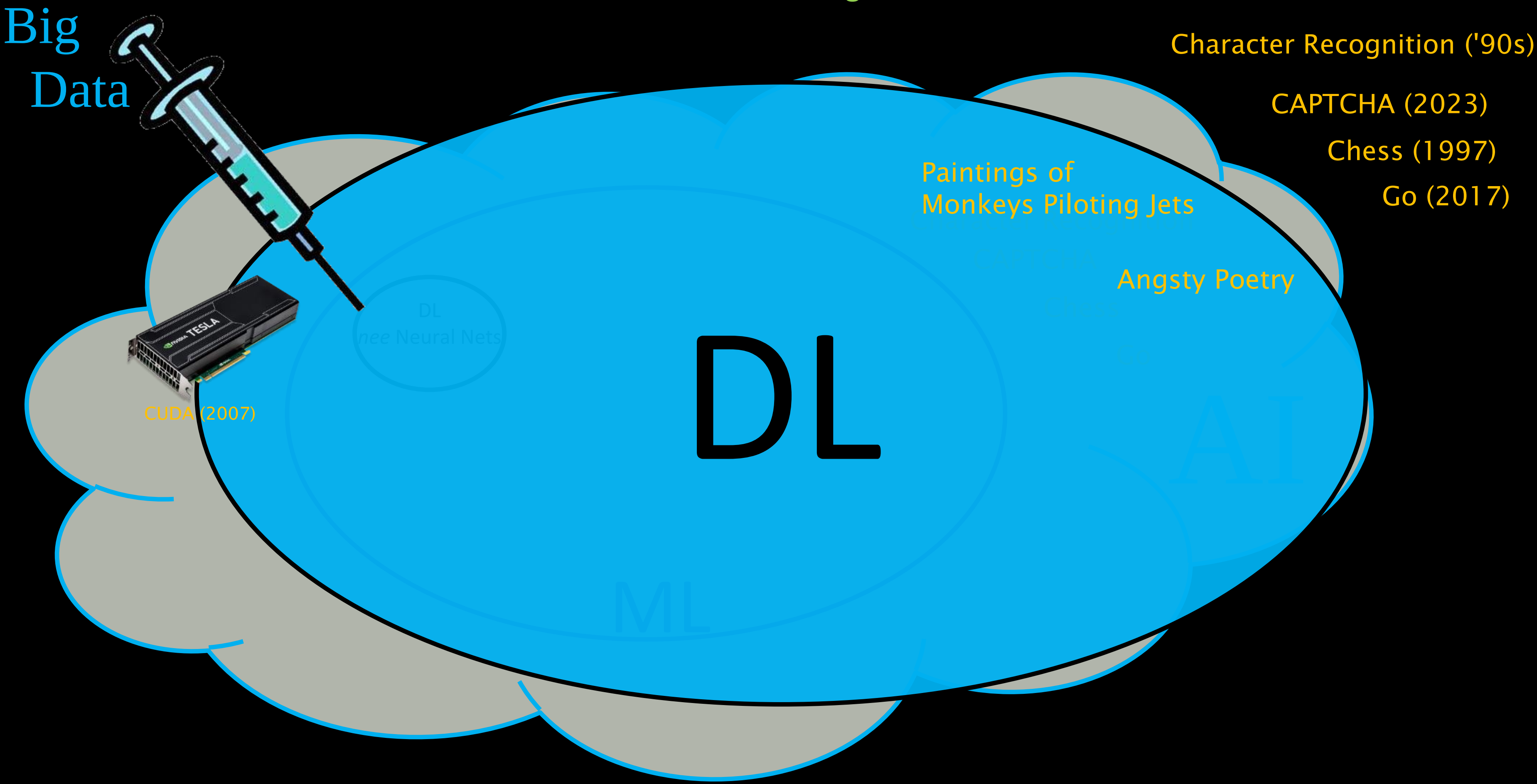

Sparse Random Projection of the digits


PCA Two Components)


t-SNE Embedding

Sample of 64-dimensional digits dataset

How does all this fit together?



Big Data

CUDA (2007)

DL

nee Neural Nets

ML

DL

AI

Character Recognition ('90s)

CAPTCHA (2023)

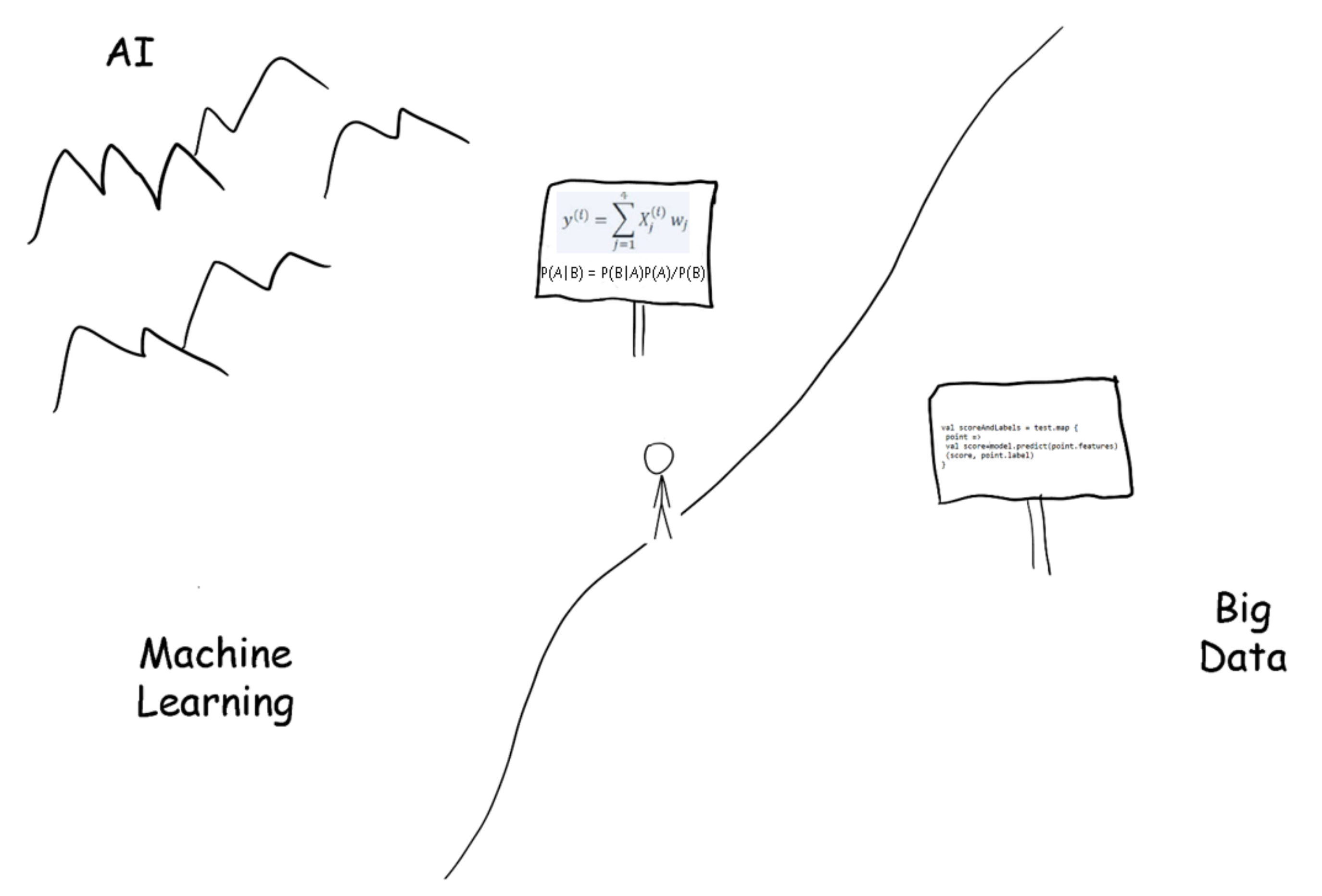
Chess (1997)

Go (2017)

Paintings of Monkeys Piloting Jets

Angsty Poetry

The Journey Ahead



$$y^{(t)} = \sum_{j=1}^{4} X_j^{(t)} w_j$$

$P(A|B) = P(B|A)P(A)/P(B)$

```
val scoreAndLabels = test.map {
 point =>
 val score=model.predict(point.features)
 (score, point.label)
}
```

As the Data Scientist wanders across the ill-defined boundary between Data Science and Machine Learning, in search of the fabled land of Artificial Intelligence, they find that the language changes from programming to a creole of linear algebra and probablity and statistics.