Intro To Big Data and Machine Learning: Hands On

Bryon Gill
Pittsburgh Supercomputing Center

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])] )
>>>
```

Read into RDD

Transform to words and integers

```
br06% interact -n 7 -R BigRM6Jul11
# wait for a job to start...
r288% module load spark
r288% pyspark


# Important: 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. Lazy
evaluation's dark side!
```

Shakespeare, a Data Analytics Favorite

Applying data analytics to the works of Shakespeare has become all the rage. Whether determining the legitimacy of his authorship (it wasn't Marlowe) or if Othello is actually a comedy (perhaps), or which word makes Macbeth so creepy ("the", yes) it is amazing how much publishable research has sprung from the recent analysis of 400 year old text.

We're going to do some exercises here using a text file containing all of his works.

A modest exercise.

We have an input file, Complete _Shakespeare.txt, that you can also find at http://www.gutenberg.org/ebooks/100.
You might find it useful to have http://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.html#pyspark.RDD in a browser window.

If you are starting from scratch on the login node:

```
#Copy all of the hands-on exercises and datasets into your home directory:
cp ~training/BigData .
interact
cd BigData/Shakespeare
module load spark
pyspark

# Let the pyspark shell start up...
# >>> rdd = sc.textFile("Complete_Shakespeare.txt")
```

Let's try a few simple exercises.

1) Count the number of lines
2) Count the number of words (hint: Python "split" is a workhorse)
3) Count unique words
4) Count the occurrence of each word
5) Show the top 5 most frequent words

These last two are a bit more challenging. One approach is to think "key/value". If you go that way, think about which data should be the key and don't be afraid to swap it about with value. This is a very common manipulation when dealing with key/value organized data.

Some Simple Answers

```
>>> lines_rdd = sc.textFile("Complete_Shakespeare.txt")
>>>
>>> lines_rdd.count()
124787
>>>
>>> words_rdd = lines_rdd.flatMap(lambda x: x.split())
>>> words_rdd.count()
904061
>>>
>>> words_rdd.distinct().count()
67779
>>>
```

Next, I know I'd like to end up with a pair RDD of sorted word/count pairs:

```
(23407, 'the'), (19540,'I'), (15682, 'to'), (15649, 'of') ...
```

Why not just *words_rdd.countByValue()*? It is an *action* that gives us a massive Python unsorted dictionary of results:

```
   ... 1, 'precious-princely': 1, 'christenings?': 1, 'empire': 11, 'vaunts': 2, 'Lubber's': 1,
'poet.': 2, 'Toad!': 1, 'leaden': 15, 'captains': 1, 'leaf': 9, 'Barnes,': 1, 'lead': 101, 'Hell':
 1, 'wheat,': 3, 'lean': 28, 'Toad,': 1, 'trencher!': 2, '1.F.2.': 1, 'leas': 2, 'leap': 17, ...
```

Where to go next? Sort this in Python or try to get back into an RDD? If this is truly *BIG* data, we want to remain as an RDD until we reach our final results. So, no.

Some Harder Answers

```
>>> lines_rdd = sc.textFile("Complete_Shakespeare.txt")
>>>
>>> lines_rdd.count()
124787
>>>
>>> words_rdd = lines_rdd.flatMap(lambda x:
>>> words_rdd.count()
904061
>>>
>>> words_rdd.distinct().count()
67779
>>>
>>> key_value_rdd = words_rdd.map(lambda x: (x,1))
>>>
>>> key_value_rdd.take(5)
[('The', 1), ('Project', 1), ('Gutenberg', 1), ('EBook', 1), ('of', 1)]
>>>
>>> word_counts_rdd = key_value_rdd.reduceByKey(lambda x,y: x+y)
>>> word_counts_rdd.take(5)
[('fawn', 11), ('considered-', 1), ('Fame,', 3), ('mustachio', 1), ('protested,', 1)]
>>>
>>> flipped_rdd = word_counts_rdd.map(lambda x: (x[1],x[0]))
>>> flipped_rdd.take(5)
[(11, 'fawn'), (1, 'considered-'), (3, 'Fame,'), (1, 'mustachio'), (1, 'protested,')]
>>>
>>> results_rdd = flipped_rdd.sortByKey(False)
>>> results_rdd.take(5)
[(23407, 'the'), (19540, 'I'), (18358, 'and'), (15682, 'to'), (15649, 'of')]
>>>
```

Things data scientists do.

Turn these into k/v pairs

Reduce to get words counts

Flip keys and values so we can sort on wordcount instead of words.

```
results_rdd = lines_rdd.flatMap(lambda x: x.split()).map(lambda x: (x,1)).reduceByKey(lambda x,y: x+y).map(lambda x: (x[1],x[0])).sortByKey(False)
```

Some Homework Problems for the ambitious.

To do research-level text analysis, we generally want to clean up our input. Here are some of the kinds of things you could do to get a more meaningful distinct word count.

1) Remove punctuation. Often punctuation is just noise, and it is here. Do a Map and/or Filter (some punctuation is attached to words, and some is not) to eliminate all punctuation from our Shakespeare data. Note that if you are familiar with regular expressions, Python has a ready method to use those.

2) Remove stop words. Stop words are common words that are also often uninteresting ("I", "the", "a"). You can remove many obvious stop words with a list of your own, and the *MLlib* that we are about to investigate has a convenient *StopWordsRemover()* method with default lists for various languages.

3) Stemming. Recognizing that various different words share the same root ("run", "running") is important, but not so easy to do simply. Once again, Spark brings powerful libraries into the mix to help. A popular one is the Natural Language Tool Kit. You should look at the docs, but you can give it a quick test quite easily:

```
import nltk
from nltk.stem.porter import  *
stemmer = PorterStemmer()
stems_rdd = words_rdd.map( lambda x: stemmer.stem(x) )
```

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.

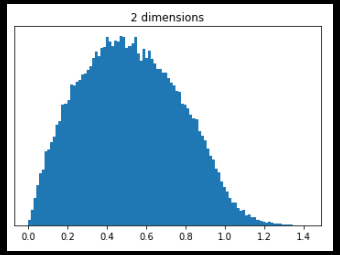
Customer Representation									
Cluster	Young Parents	College Athlete	Auto Enthusiast	Knitter	Steelers Fan	Shakespeare Reader	Sci-Fi Fan	Plumber	...
Distance	0.02	2.3	1.4	8.4	2.2	14.9	3.3	0.8	...

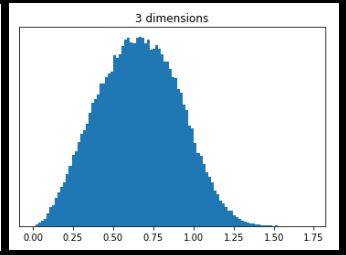
We have now accomplished two things:

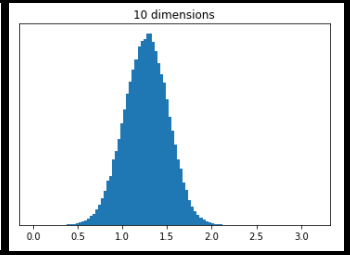
- we have compressed our data
- learned something about our customers (who to send a dashcam promo to).

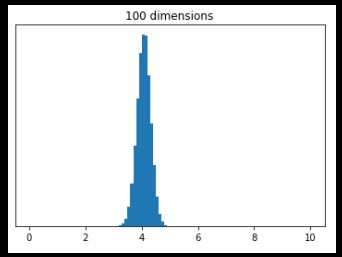
Curse of Dimensionality

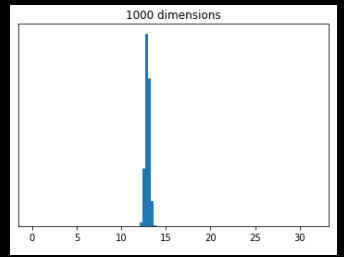
This is a good time to point out how our intuition can lead us astray as we increase the dimensionality of our problems - which we will certainly be doing - and to a great degree. There are several related aspects to this phenomenon, often referred to as the *Curse of Dimensionality*. One root cause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

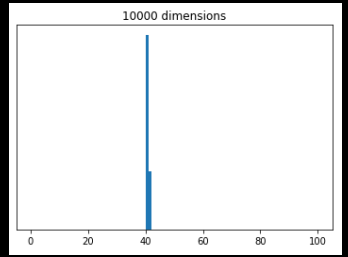












These plots show the distributions of pairwise distances between randomly distributed points within differently dimensioned unit hypercubes. Notice how all the points start to be about the same distance apart.

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.

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.

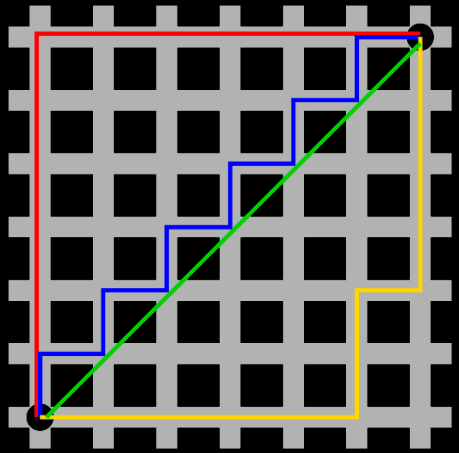


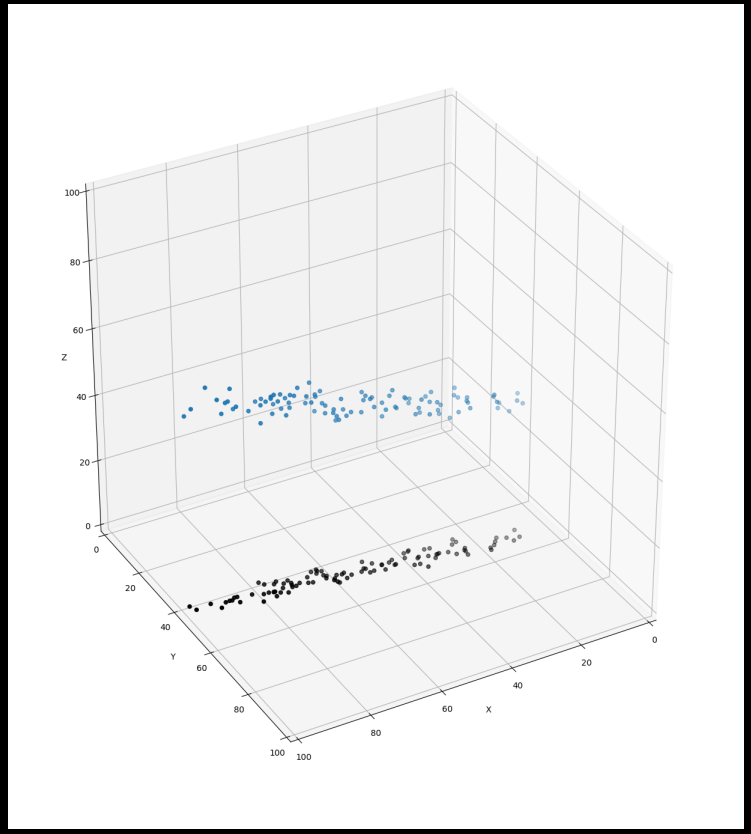
Image Source: Wikipedia

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.

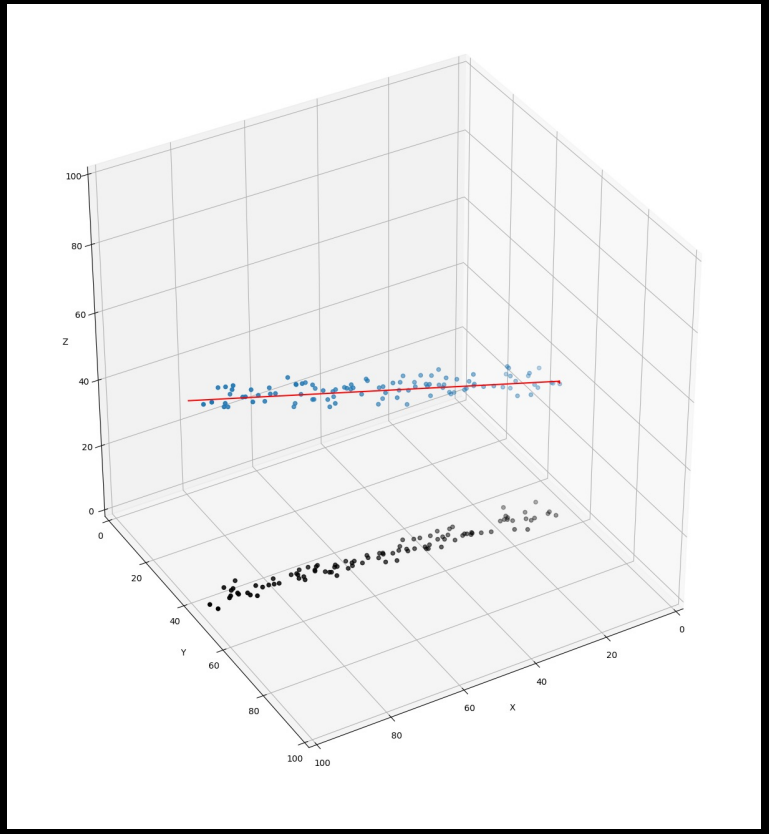
For genomics, you might care more about string sequences.

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

Alternative DR: Principal Component Analysis

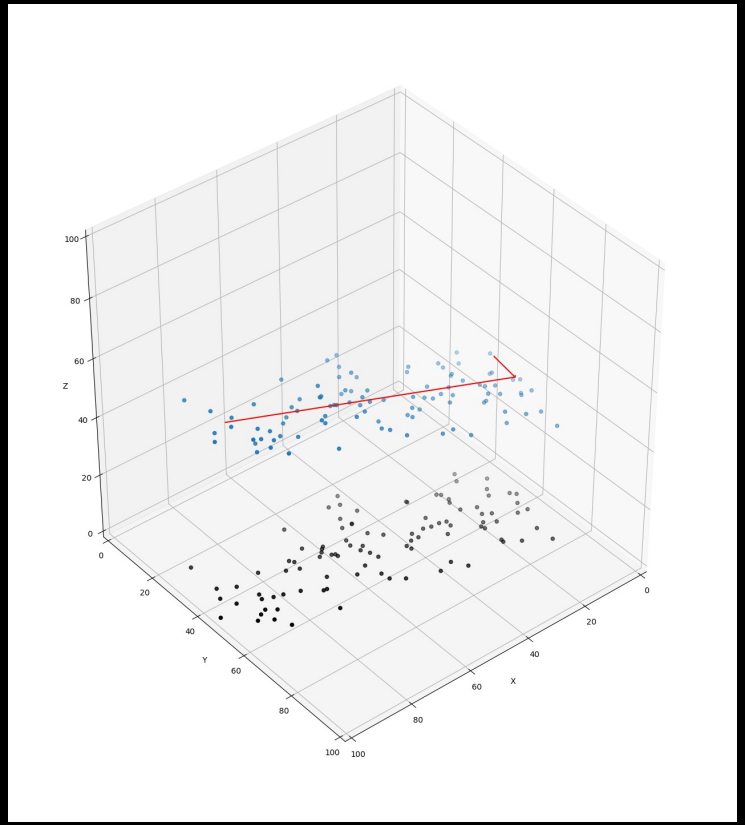


3D Data Set

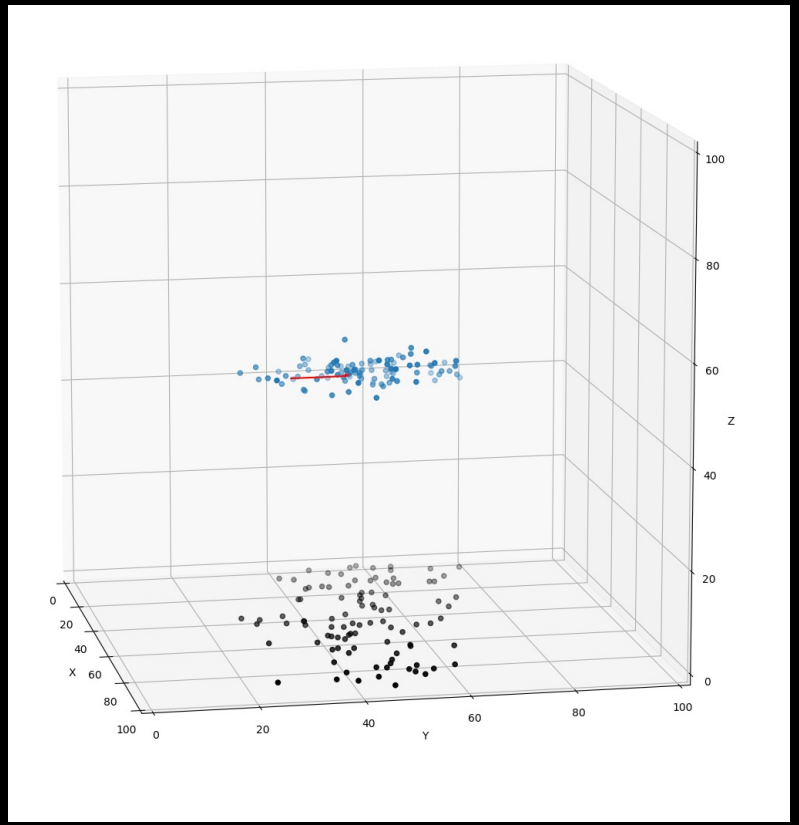


Maybe mostly 1D!

Alternative DR: Principal Component Analysis



Flatter 2D-ish Data Set



View down the 1st Princ. Comp.

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:

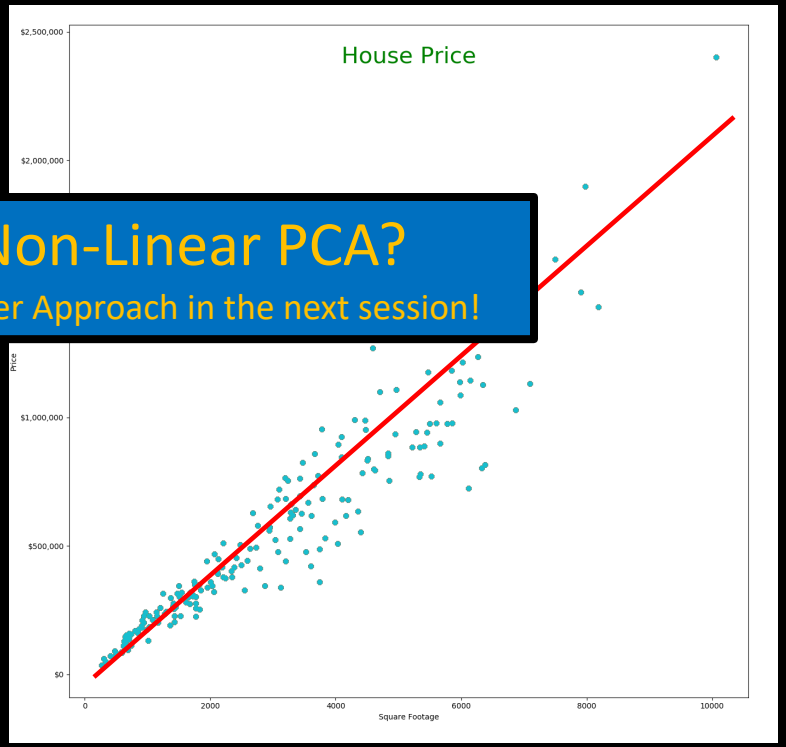
Home Data	
Latitude	4833438 north
Longitude	630084 east
Last Sale Price	$ 480,000
Last Sale Year	1998
Width	62
Depth	40
Floors	3
Bedrooms	3
Bathrooms	2
Garage	2
Yard Width	84
Yard Depth	60
...	...

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

Principal Component Analysis Fail

1st Component Off

Data Not Very Linear



D x W Is Not Linear

But (DxW) Fits Well

Non-Linear PCA?

A Better Approach in the next session!

Why the fascination with linear techniques?

The Streetlight Effect

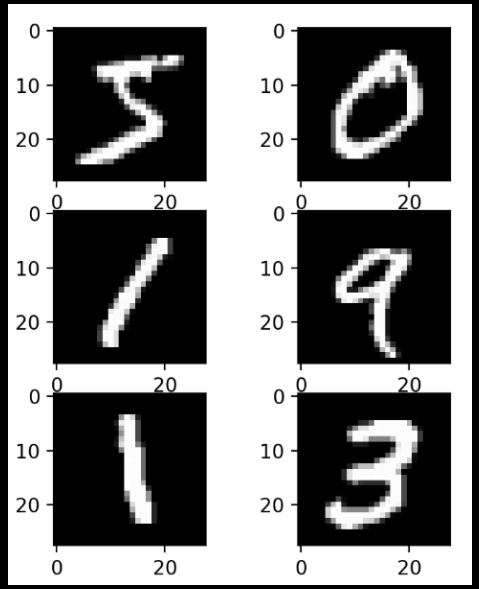
This is a very real and powerful force throughout the sciences.

It is not because practitioners are dumb.

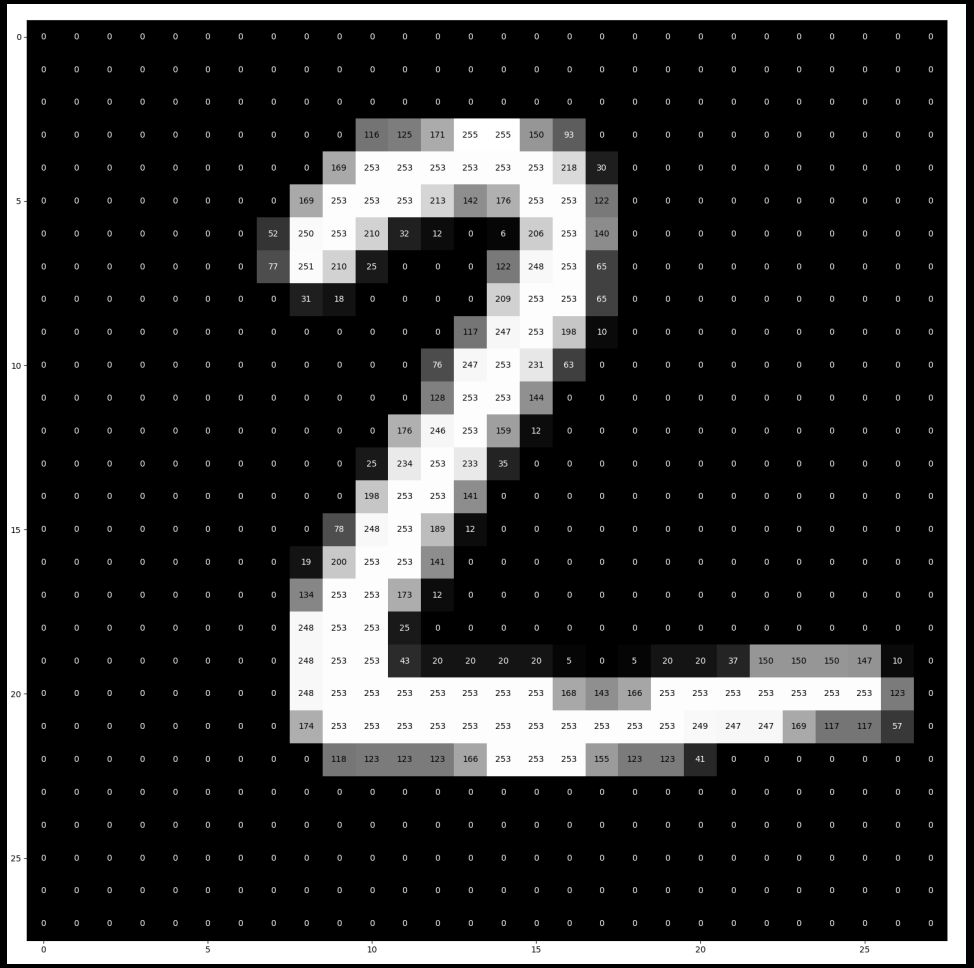
But, it is also very often neither explained nor justified.

Which leads to great confusion.

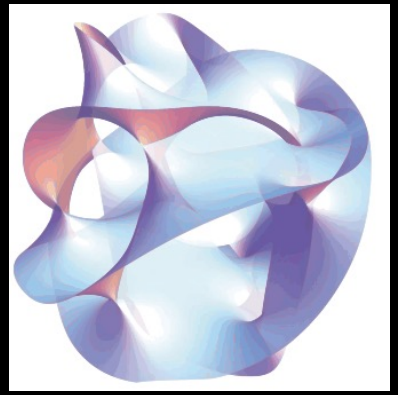
Why Would An Image Have 784 Dimensions?



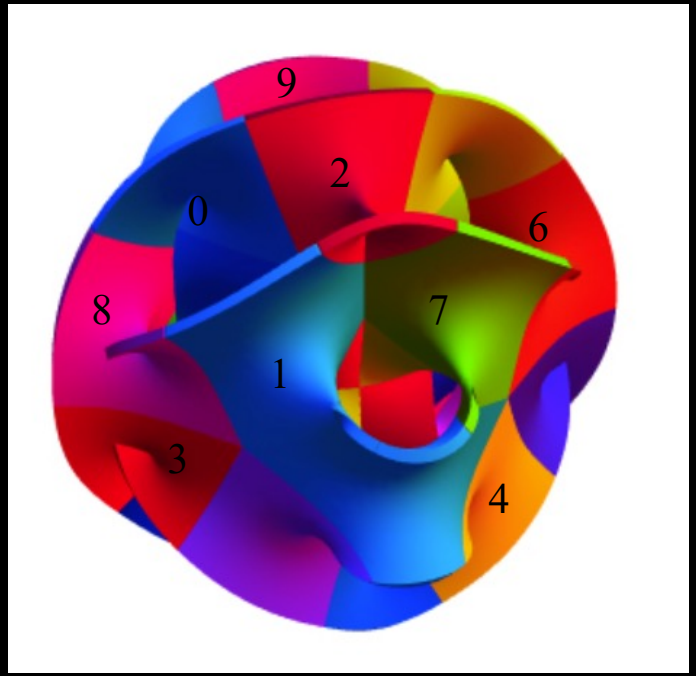
MNIST 28x28
greyscale images



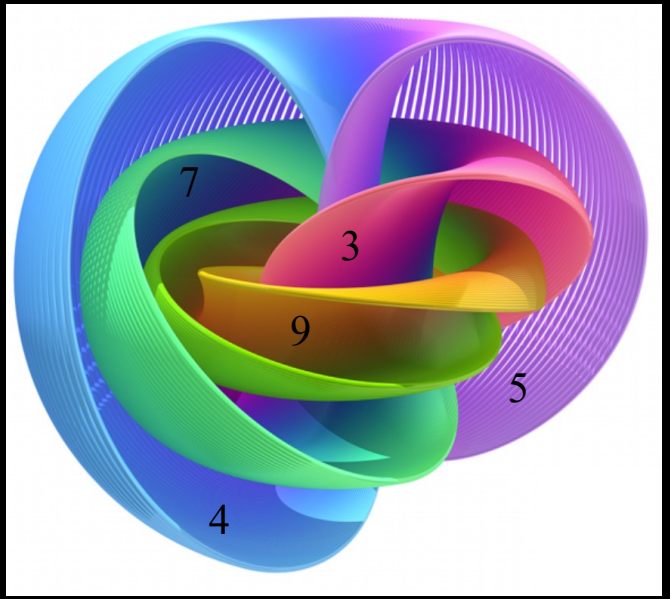
Central Hypothesis of Modern DL



Data Lives On
A Lower Dimensional
Manifold



Maybe Very Contiguous

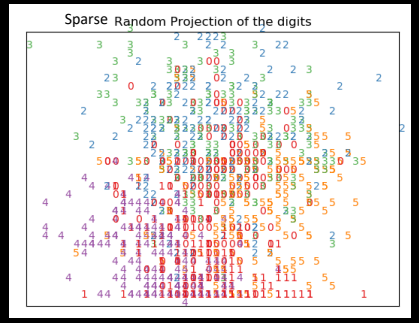


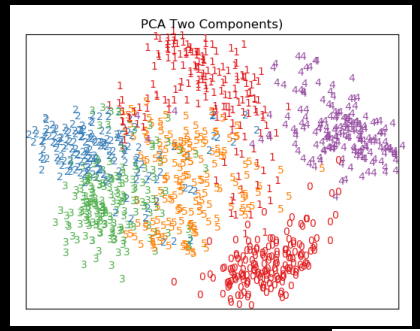
Maybe A Small Set
Of Disconnected

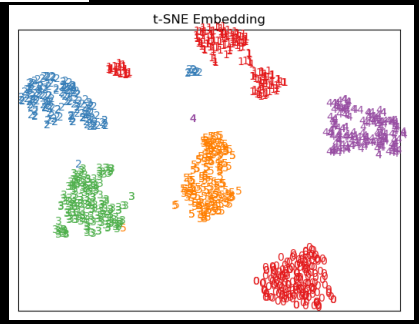
Images from Wikipedia

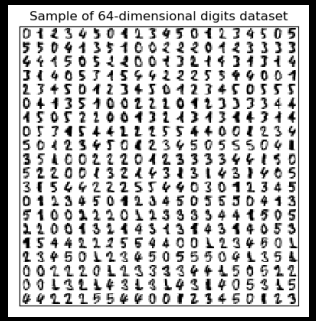
Testing These Ideas With Scikit-learn

```
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()
```









How does all this fit together?

Character Recognition
Capchas
Chess
Go
Paintings of
Monkeys Piloting Jets
Angsty Poetry

DL
nee Neural Nets

DL

ML

AI