Machine Learning 1.10: Curse of Dimensionality

Tom S. F. Haines
T.S.F.Haines@bath.ac.uk

Curse of Dimensionality

- Given a name by Richard Bellman
  (“Adaptive Control Processes: A Guided Tour”, 1961)
  (Inventor of dynamic programming – next lecture)

- Short version: More dimensions $\implies$ need more data
  (dimension = feature)

Curse of Dimensionality

- Given a name by Richard Bellman
  ("Adaptive Control Processes: A Guided Tour", 1961)
  (Inventor of dynamic programming – next lecture)
- Short version: More dimensions $\implies$ need more data
  (dimension = feature)
- Why?
- How much more?

Combinatorics

- Imagine a problem:
  - Feature vector of length n
  - Binary – answers to *yes*/*no* questions
  - Any output

  Typical of surveys and psychological test.

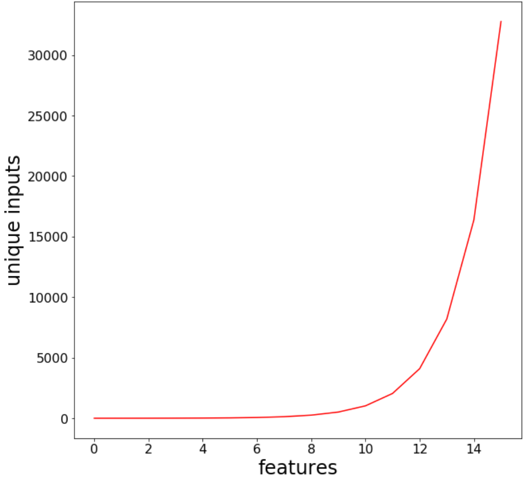
Combinatorics

- Imagine a problem:
  - Feature vector of length n
  - Binary – answers to *yes*/*no* questions
  - Any output

  Typical of surveys and psychological test.

- There are 2^n possible inputs
- If you have 1000 unique exemplars:

n	coverage
10	100.0%
50	40.0%
100	10.0%
500	0.4%
1000	0.1%



Coverage

- Machine learning is built on similarity,
  the assumption that a similar feature vector implies a similar answer

- Low coverage $\implies$ nothing is similar

Coverage

- Machine learning is built on similarity,
  the assumption that a similar feature vector implies a similar answer
- Low coverage $\implies$ nothing is similar
- There is still something "most similar" and "least similar" however...

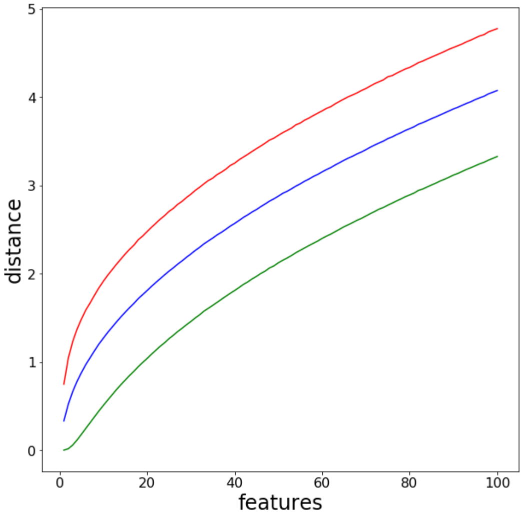
Distance in $100D$ Space

- 1000 points in nD space, uniform distribution, $x \in [0, 1]^n$
- How far away is everything? (Euclidean)

Distance in $100D$ Space

- 1000 points in nD space, uniform distribution, $x \in [0, 1]^n$
- How far away is everything? (Euclidean)

- Right: Typical values
  Red = maximum
  Blue = mean
  Green = minimum



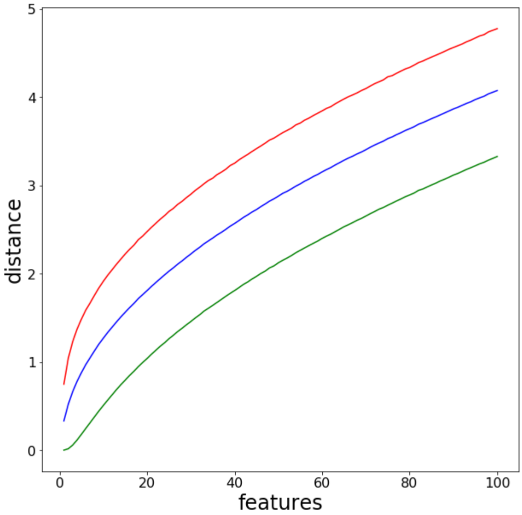
Distance in $100D$ Space

- 1000 points in nD space, uniform distribution, $x \in [0,1]^n$
- How far away is everything? (Euclidean)

- Right: Typical values
  Red = maximum
  Blue = mean
  Green = minimum

- Distance increases, range does not.
- $\frac{\texttt{maximum}}{\texttt{minimum}}$ decreases.

- Everything starts to look the same!



Never Enough

- Poor coverage = failing similarity
- However much data you have, the curse always gets you

Never Enough

- Poor coverage = failing similarity
- However much data you have, the curse always gets you

- Consider nearest neighbours (set point to same class as nearest)
- First dimension, x[0]:

  $x[0] < \xrightarrow{1/3}$ Class 1,

  $x[0] > \xrightarrow{2/3}$ Class 2

  (gap between empty)
- All further dimensions are noise, $\sim \text{Normal}(0, 1)$

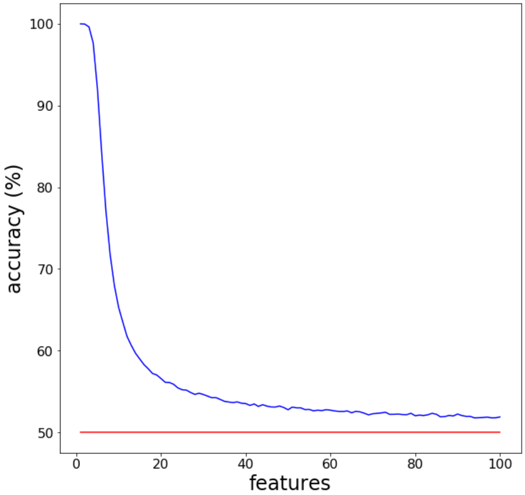
Never Enough

- Poor coverage = failing similarity
- However much data you have, the curse always gets you

- Consider nearest neighbours (set point to same class as nearest)
- First dimension, x[0]:

  $$x[0] < \xrightarrow{1/3} \text{Class 1,}$$
  $$x[0] > \xrightarrow{2/3} \text{Class 2}$$
  (gap between empty)
- All further dimensions are noise, $\sim \text{Normal}(0, 1)$

- This should be easy, but...



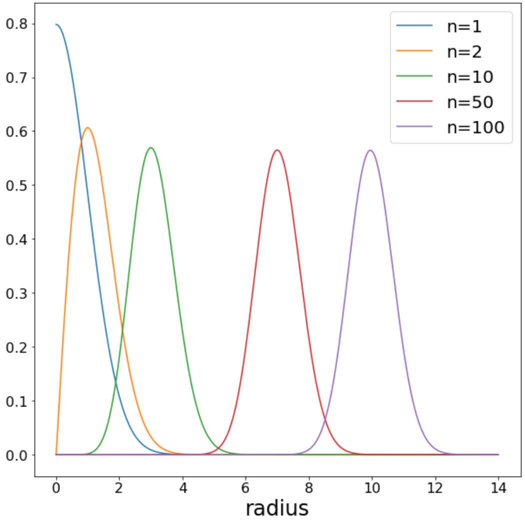
Not the Distributions

- It also affects probability distributions
  e.g. the standard multivariate Gaussian

Not the Distributions

- It also affects probability distributions e.g. the standard multivariate Gaussian

- Consider draws from the distribution
- Calculate the distribution of their distances from the mean (called a χ (chi) distribution)

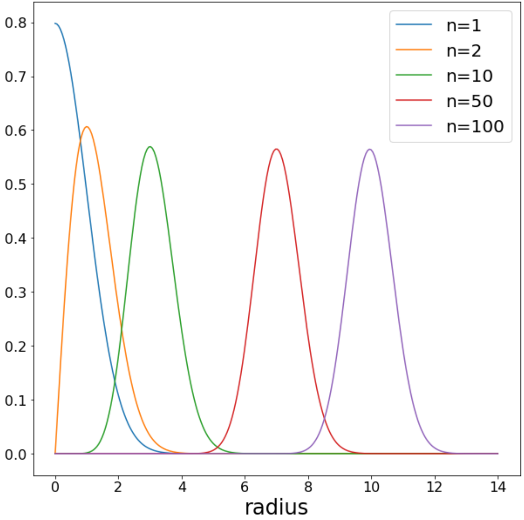


Not the Distributions

- It also affects probability distributions e.g. the standard multivariate Gaussian

- Consider draws from the distribution
- Calculate the distribution of their distances from the mean (called a χ (chi) distribution)

- In high dimensions most draws are in a thin shell, nowhere near the mean!



Solutions

- Feature selection
- Feature engineering

- Problem doesn't exist for real data! (manifolds)

Feature Selection

- Select a small number of features to use, ignore rest
- Three approaches:
  - **Model Selection**: Train model with many subsets of the features and select best
  - **Filtering**: Use an estimate of feature usefulness and filter accordingly
  - **Embedded**: An algorithm with feature selection built in

- Poor relative of structure learning – later lecture

Feature Selection: Model Selection

- Clearly the best approach, but slow
- Impractical to search exhaustively

Feature Selection: Model Selection

- Clearly the best approach, but slow
- Impractical to search exhaustively

- Typical approach:
  1. Start with empty set
  2. Consider n modifications: Adding (not in set) or removing (in set) each variable
  3. Select the best; usually with an extra cost on number of features used
  4. If best changed return to step 2, otherwise exit

Feature Selection: Model Selection

- Clearly the best approach, but slow
- Impractical to search exhaustively

- Typical approach:
  1. Start with empty set
  2. Consider *n* modifications: Adding (not in set) or removing (in set) each variable
  3. Select the best; usually with an extra cost on number of features used
  4. If best changed return to step 2, otherwise exit

- **Stepwise regression**: The above with a regression model, e.g. Logistic regression
- Possibility that considering larger “moves” would find a better solution

Feature Selection: Filtering

- Questionable approach, but fast:
  1. Calculate how useful every feature is
  2. Select set based on a threshold
  3. Train model

Feature Selection: Filtering

- Questionable approach, but fast:
  1. Calculate how useful every feature is
  2. Select set based on a threshold
  3. Train model

- Example: Threshold absolute correlation

Feature Selection: Filtering

- Questionable approach, but fast:
  1. Calculate how useful every feature is
  2. Select set based on a threshold
  3. Train model

- Example: Threshold absolute correlation

- "Usefulness" is dependent on other variables:
  - Two variables might contain the same information $\implies$ including both is pointless
  - A variable might be useless on it's own, but really helpful with others

  There are variants that consider such relationships

Feature Selection: Embedded

- Good trade off of performance/speed
- Example: Random forests – Feature/split selection is feature filtering

Feature Selection: Embedded

- Good trade off of performance/speed
- Example: Random forests – Feature/split selection is feature filtering

- Random forest can have an infinite number of features!
- Example:

  Kinect person tracker labels pixels – left forearm, head, background...
  Generates pairs of random offsets from current pixel
  Splits based on relative depth between pixel at offsets
  (perspective is factored in – offsets are in meters)

Feature Engineering

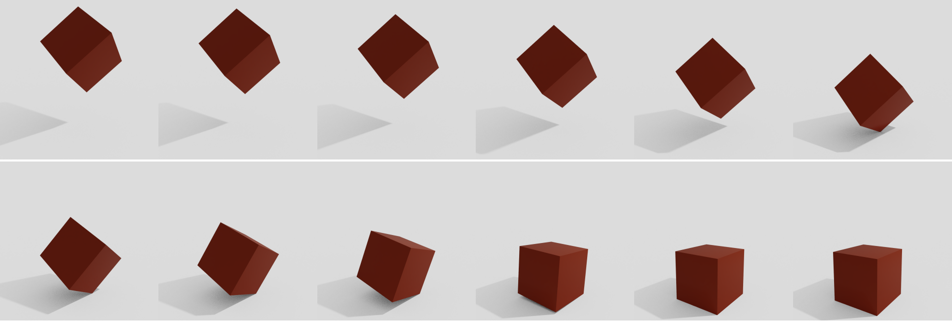
- Use domain knowledge to design new features
- Small number, based on what you think will work
  (use data exploration to help)

Feature Engineering

- Use domain knowledge to design new features
- Small number, based on what you think will work
  (use data exploration to help)
- Example: Radius trick to fit a circle with Logistic regression
- Invariance and equivariance

Data in the Real World

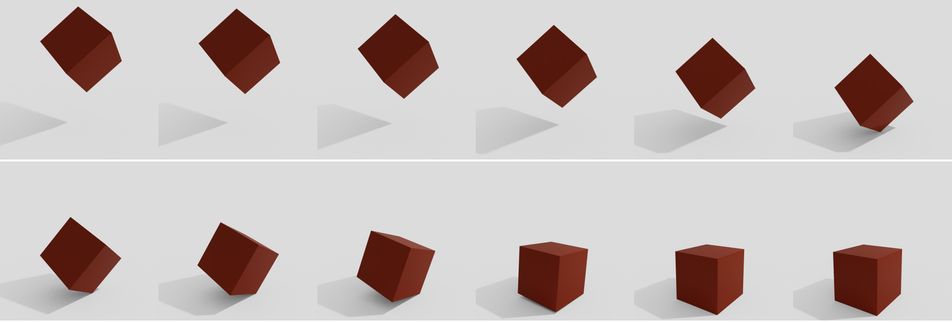
Consider a video of a cube falling:



- How many dimensions does each frame have?

Data in the Real World

Consider a video of a cube falling:



- How many dimensions does each frame have?

- Formally: 3145728: 1024×1024 images, 3 channels per pixel.
- Actually: 6: 3 for position of cube, 3 for orientation of cube. (everything else is a (complex) function of these 6 parameters)

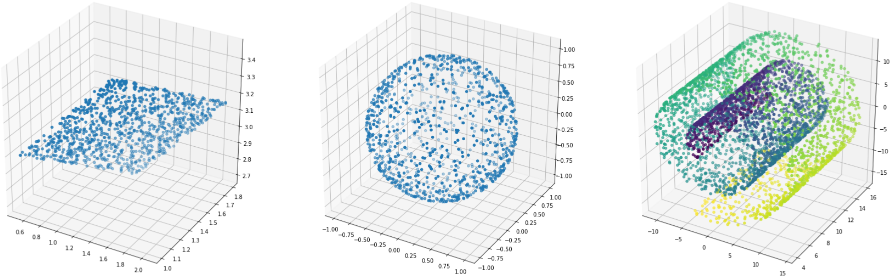
- **Real data is mostly low dimensional**!

Manifolds

- A manifold is the low dimensional surface the data is actually on.

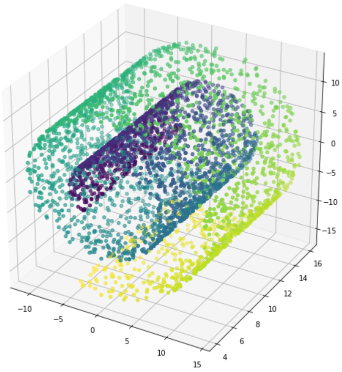
Manifolds

- A manifold is the low dimensional surface the data is actually on.
- $2D$ manifolds embedded in $3D$ space:

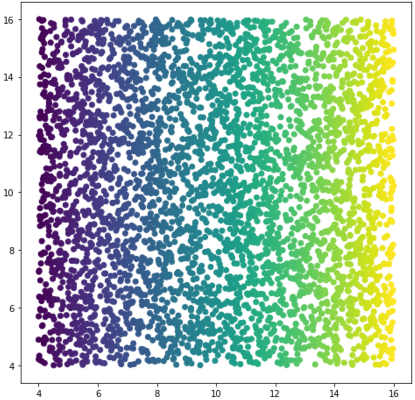


Dimensionality Reduction I

- Assign a coordinate system to the manifold, use that instead:



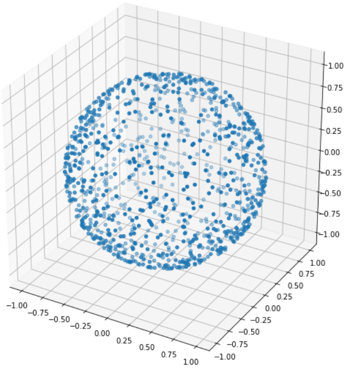
→



- Often used for visualisation (you used PCA previously)

Dimensionality Reduction II

- Doesn't work if you have loops however!



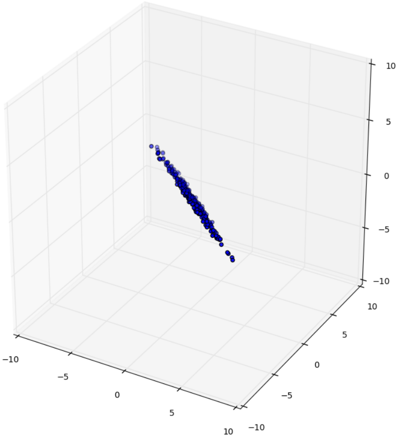
Principal Component Analysis

- Invented in 1901 by Karl Pearson!
  "On Lines and Planes of Closest Fit to Systems of Points in Space"

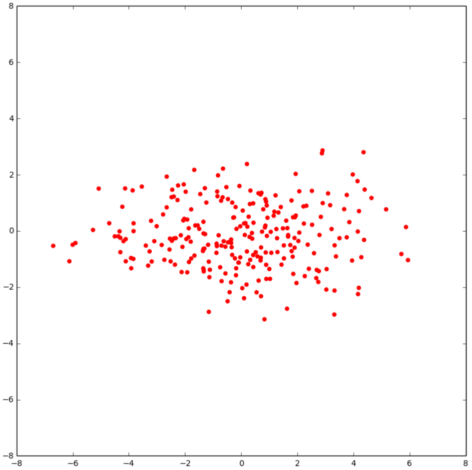
- Really simple
- Great for visualisation and dimensionality reduction

- Only handles linear manifolds (e.g. a $4D$ hyper-cube embedded in $12D$ space)
- But still useful, even for non-linear manifolds

Example



$\longrightarrow$



One Feature

- Imagine we have to represent a nD data set using one feature!
- How do we preserve the most information?

One Feature

- Imagine we have to represent a *nD* data set using one feature!
- How do we preserve the most information?

- Don't have to select one feature – can take a linear combination
  e.g. $\text{keep} = 0.2x_0 - 0.3x_1 + 0.1x_2$

One Feature

- Imagine we have to represent a nD data set using one feature!
- How do we preserve the most information?

- Don't have to select one feature – can take a linear combination
  e.g. keep $= 0.2x_0 - 0.3x_1 + 0.1x_2$

- Scale of linear combination does not matter, so assume
  keep $= \hat{d} \cdot x$, $|\hat{d}| = 1$

One Feature

- Imagine we have to represent a nD data set using one feature!
- How do we preserve the most information?

- Don't have to select one feature – can take a linear combination
  e.g. $\text{keep} = 0.2x_0 - 0.3x_1 + 0.1x_2$

- Scale of linear combination does not matter, so assume
  $\text{keep} = \hat{d} \cdot x$, $|\hat{d}| = 1$

- The most information is preserved if we select the direction of maximum variance.

Maximum Variance

- Whiteboard!

- Note: Always passes through mean
- Obtains first principal component

More Principal Components

- Repeat process, for remaining information
- That is, each new principal component must be orthogonal to all previous
- Orthogonality means no shared information

- In practise, there is an analytic solution

Algorithm

- Subtract mean
- Calculate eigenvectors and eigenvalues of covariance matrix
- Energy is the sum of eigenvalues
- Keep the eigenvectors associated with the largest eigenvalues, to obtain 99.9% of energy (or another percentage, or $2/3$ if for visualisation)
- Transform data with matrix of kept eigenvectors

- Can return to original space using transpose of kept eigenvectors

UNIVERSITY OF BATH

Python

```
data -= data.mean(axis=1)[:,None]
covar = numpy.cov(data.T) # Symmetric

evals, evecs = numpy.linalg.eigh(covar)
# evals is ordered lowest to highest, with evecs[:,i] matching evals[i]

energy = numpy.cumsum(evals)
start = 0
while energy[start] < 0.001 * energy[-1]: # Keep 99.9% of energy
    start += 1

project = evecs[:,start:][:,::-1].T
projected = project.dot(data.T).T
```

Further Manifold Approaches

- Iso-map – non-linear, much better than PCA for visualisation
- "Learning a Manifold as an Atlas" by N. Pitelis, C. Russell and L. Agapito – non-linear by segmenting manifold into sections where PCA (linear) can be applied

Future Curse-avoiding Approaches

Approaches that will be covered latter:

- Learning features with convolutional NN – next term
- Using structure (conditional independence) – next lecture

Summary

- The curse of dimensionality
  (beware your intuitions in high dimensions)

- Feature selection
- Feature engineering
- Manifolds & dimensionality reduction

Further Reading

- "A Few Useful Things to Know about Machine Learning", by P. Domingos, `https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf`