# Machine Learning 1.10: The Curse of Dimensionality

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

UNIVERSITY OF BATH

# Curse of dimensionality

- Named by Richard Bellman
  ("Adaptive Control Processes: A Guided Tour", 1961)
  (Inventor of dynamic programming – later lecture)

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

UNIVERSITY OF BATH

# Curse of dimensionality

- Named by Richard Bellman
  ("Adaptive Control Processes: A Guided Tour", 1961)
  (Inventor of dynamic programming – later 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.

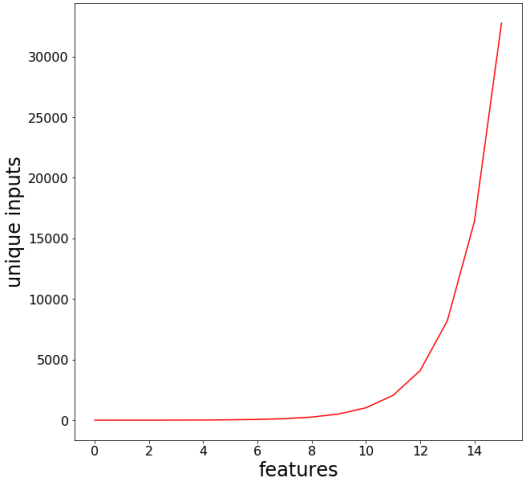
# Combinatorics

- 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% |



UNIVERSITY OF BATH

# Coverage

- Machine learning assumption:
  Similar feature vector $\implies$ similar answer
- But:
  Low coverage $\implies$ nothing is similar

# Coverage

- Machine learning assumption:
  Similar feature vector $\implies$ similar answer
- But:
  Low coverage $\implies$ nothing is similar
- There is still something "most similar" and "least similar" however...

UNIVERSITY OF BATH

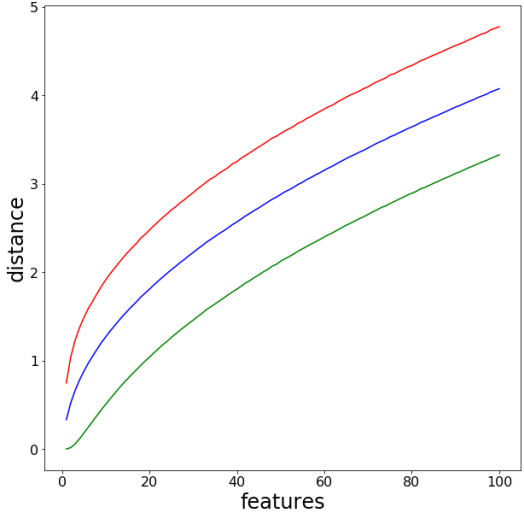
# Distance in $100D$ space

- 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)

- Expected values: (right)
  Red = maximum
  Blue = mean
  Green = minimum



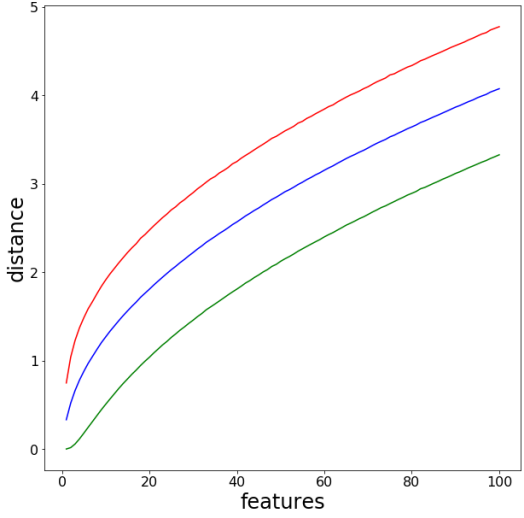
# Distance in $100D$ space

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

- Expected values: (right)
  Red = maximum
  Blue = mean
  Green = minimum

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

- Everything starts to look the same!



UNIVERSITY OF BATH

# Never enough

- Poor coverage = failing similarity
- However much data you have, the curse always gets you (can't beat $2^n$)

UNIVERSITY OF BATH

# Never enough

- Poor coverage = failing similarity
- However much data you have, the curse always gets you (can't beat $2^n$)

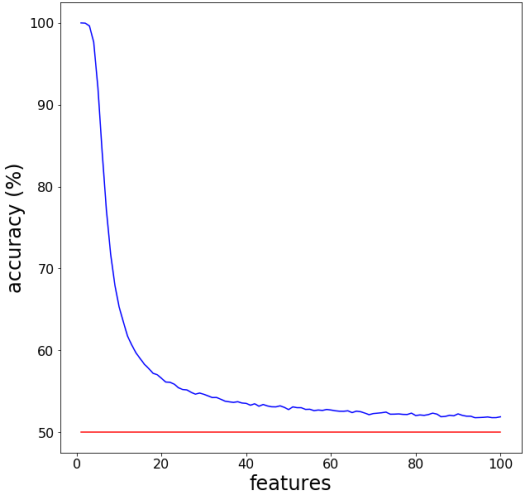
- Consider nearest neighbours (set point to same class as nearest)
- First dimension, x[0]:
  $x[0] < \frac{1}{3} \implies$ class 0,
  $x[0] > \frac{2}{3} \implies$ class 1
  (gap between empty)
- All further dimensions are noise, $\sim$ Normal$(0, 1)$

# Never enough

- Poor coverage = failing similarity
- However much data you have, the curse always gets you (can't beat $2^n$)

- Consider nearest neighbours (set point to same class as nearest)
- First dimension, x[0]:
  $x[0] < \frac{1}{3} \implies$ class 0,
  $x[0] > \frac{2}{3} \implies$ class 1
  (gap between empty)
- All further dimensions are noise, $\sim$ Normal$(0, 1)$

- This should be easy, but...



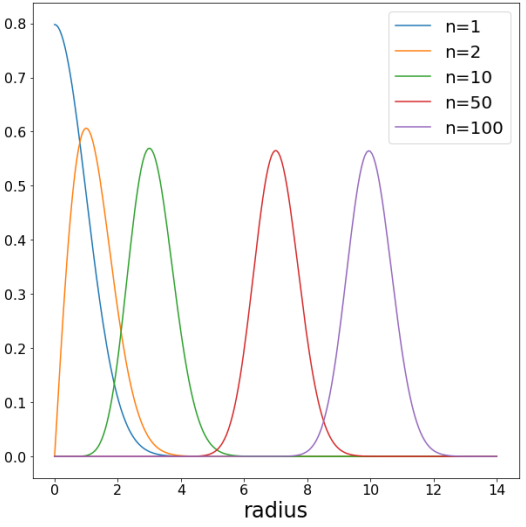
## Not the distributions as well...

- Affects probability distributions
  e.g. standard multivariate Gaussian

UNIVERSITY OF BATH

# Not the distributions as well. . .

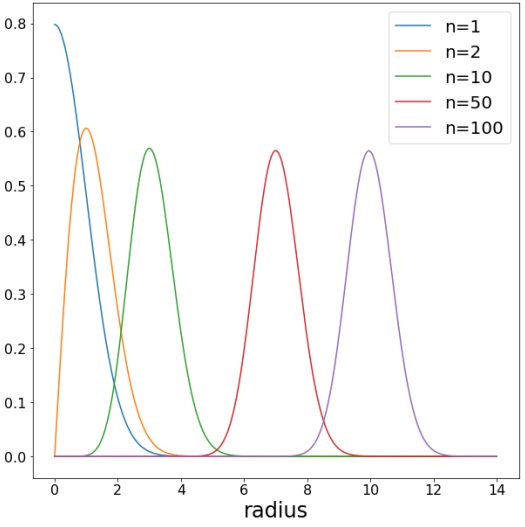
- Affects probability distributions
  e.g. standard multivariate Gaussian
- Draw samples from distribution
- Distribution of distances from mean?
  (called a $\chi$ (chi) distribution)



UNIVERSITY OF BATH

# Not the distributions as well. . .

- Affects probability distributions
  e.g. standard multivariate Gaussian
- Draw samples from distribution
- Distribution of distances from mean?
  (called a $\chi$ (chi) distribution)
- High dimensions: Samples are in a thin shell, avoiding the mean!



UNIVERSITY OF BATH

# Avoiding the curse

- Affects all models
- Solutions:
  - Feature selection
  - Feature engineering
  - Problem doesn't exist for real data!
    (manifolds; which is not to say it's just going to solve itself)
- Related to regularisation

UNIVERSITY OF BATH

# 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
- Using problem structure is better – later lectures

UNIVERSITY OF BATH

# Feature selection: Model selection

- Train model with all subsets of features, select best

- Clearly best approach
- Impractical to search exhaustively

UNIVERSITY OF BATH

# Feature selection: Model selection

- Train model with all subsets of features, select best
- Clearly best approach
- Impractical to search exhaustively
- Typically:
  1. Start with empty set
  2. Try $n$ modifications: Adding (not in set) or removing (in set) each variable
  3. Select best (usually with regularisation term on number of features used)
  4. If best changed return to step 2, otherwise exit

# Feature selection: Model selection

- Train model with all subsets of features, select best

- Clearly best approach
- Impractical to search exhaustively

- Typically:
  1. Start with empty set
  2. Try $n$ modifications: Adding (not in set) or removing (in set) each variable
  3. Select best (usually with regularisation term on number of features used)
  4. If best changed return to step 2, otherwise exit

- **Stepwise regression**: Above with a regression model, e.g. Logistic regression
- Gets stuck in local minima

UNIVERSITY OF BATH

# Feature selection: Filtering

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

UNIVERSITY OF BATH

# Feature selection: Filtering

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

- Typically: Threshold absolute correlation

$$\left| \frac{\mathbb{E}\left[(x_i - \mu_{x_i})(y - \mu_y)\right]}{\sqrt{\mathbb{E}\left[(x_i - \mu_{x_i})^2\right]\mathbb{E}\left[(y - \mu_y)^2\right]}} \right| > t$$

(often adjust threshold to achieve target feature count)

# Feature selection: Filtering

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

- Typically: Threshold absolute correlation

$$\left| \frac{\mathbb{E}\left[(x_i - \mu_{x_i})(y - \mu_y)\right]}{\sqrt{\mathbb{E}\left[(x_i - \mu_{x_i})^2\right]\mathbb{E}\left[(y - \mu_y)^2\right]}} \right| > t$$

(often adjust threshold to achieve target feature count)

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

- Too crude (linear) for interesting problems
- Other correlation metrics...

UNIVERSITY OF BATH

# Feature selection: Embedded

- Algorithm does it as part of learning
- Good trade off between performance/speed

- Best example: Random forests
  - Feature/split selection is feature filtering
  - Significant part of why they perform so well

UNIVERSITY OF BATH

# Feature engineering

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

UNIVERSITY OF BATH

# 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
- Often about **invariance**

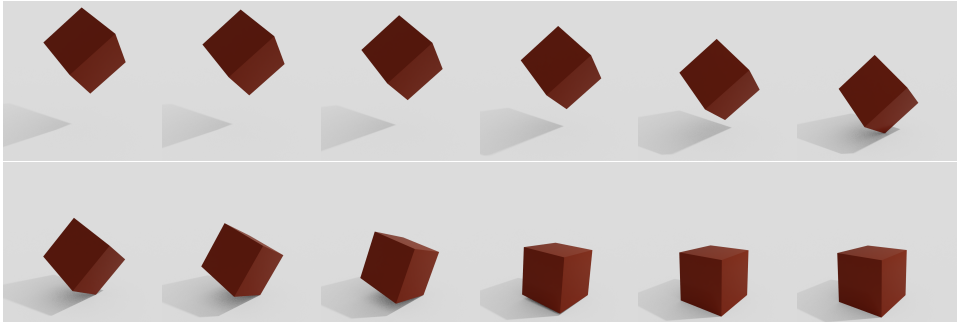
UNIVERSITY OF BATH

# 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
- Often about **invariance**
- Can massively improve performance...
  ...but requires domain knowledge

# Real data

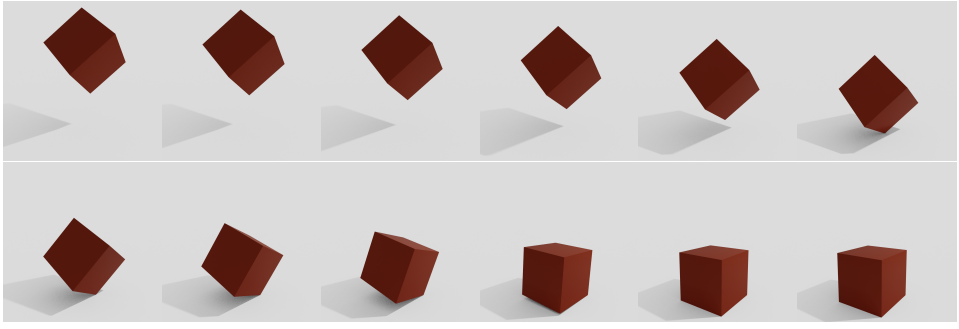
Consider a video of a cube falling:



- How many dimensions does each frame have?

# Real data

Consider a video of a cube falling:



- How many dimensions does each frame have?
- Formally: 3145728: $1024 \times 1024$ 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**!

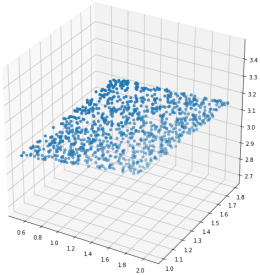
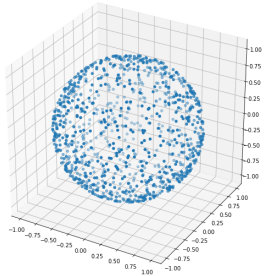
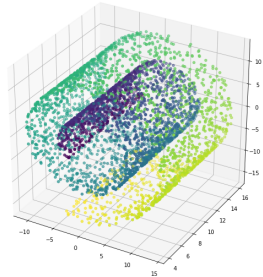
# Manifolds

- Manifold: Low dimensional surface embedded in higher dimensional space
- Real data typically lies on a manifold
  (in MI we generalise maths definition slightly: Manifold dimensionality can vary)

UNIVERSITY OF BATH

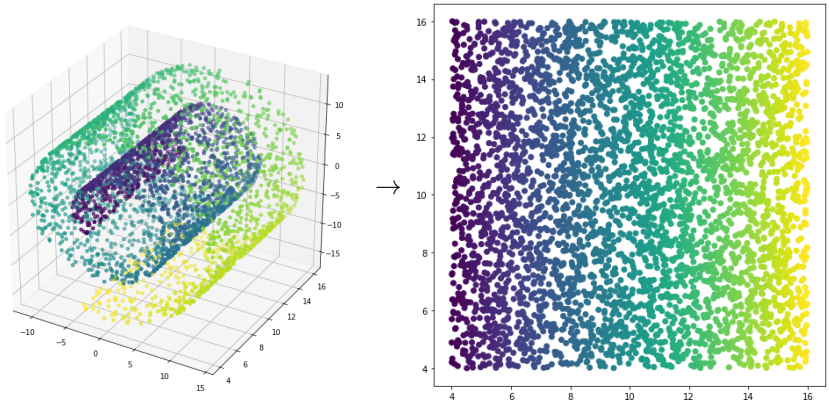
# Manifolds

- Manifold: Low dimensional surface embedded in higher dimensional space
- Real data typically lies on a manifold
  (in MI we generalise maths definition slightly: Manifold dimensionality can vary)
- e.g. $2D$ manifolds embedded in $3D$ space:

# Dimensionality reduction I

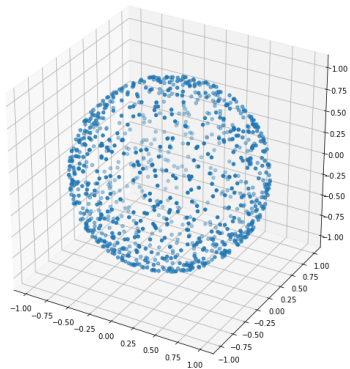
- Assign a coordinate system to the manifold, use that instead:




- Also used for visualisation (e.g. you used PCA previously)

# Dimensionality reduction II

- Doesn't work if you have loops however!



UNIVERSITY OF BATH

# Principal component analysis

- Invented in 1901 by Karl Pearson!
  "On Lines and Planes of Closest Fit to Systems of Points in Space"
- Simple, fast
- Great for visualisation and dimensionality reduction

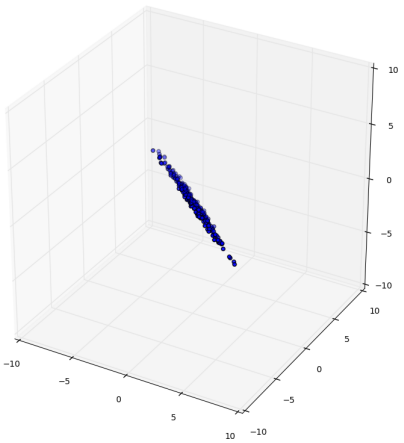
UNIVERSITY OF BATH

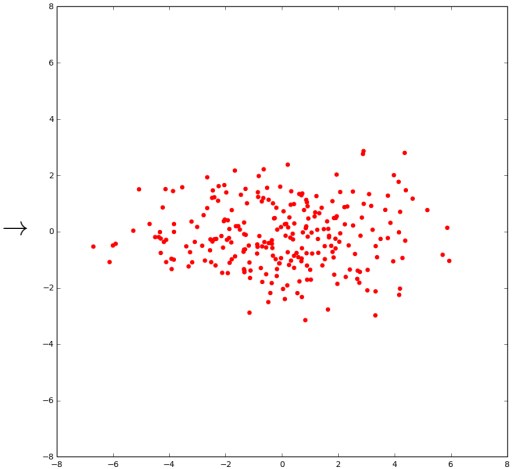
# Principal component analysis

- Invented in 1901 by Karl Pearson!
  "On Lines and Planes of Closest Fit to Systems of Points in Space"
- Simple, fast
- Great for visualisation and dimensionality reduction
- Linear manifolds only (e.g. $4D$ hyper-cube embedded in $12D$ space)
- Still valuable for non-linear manifolds
- Note: Related to factor analysis

UNIVERSITY OF BATH

# Example



# One feature

- Imagine: Represent $nD$ data set using only one feature!
- How do we preserve the most information?

# One feature

- Imagine: Represent $nD$ data set using only one feature!
- How do we preserve the most information?

- Don't have to select specific feature
- Take a linear combination instead
  e.g. keep $= 0.7x_0 - 0.6x_1 + 0.4x_2$

# One feature

- Imagine: Represent $nD$ data set using only one feature!
- How do we preserve the most information?

- Don't have to select specific feature
- Take a linear combination instead
  e.g. keep $= 0.7x_0 - 0.6x_1 + 0.4x_2$

- Scale does not matter $\therefore$ assume keep $= \hat{d} \cdot x$, $|\hat{d}| = 1$

# One feature

- Imagine: Represent $nD$ data set using only one feature!
- How do we preserve the most information?

- Don't have to select specific feature
- Take a linear combination instead
  e.g. keep $= 0.7x_0 - 0.6x_1 + 0.4x_2$

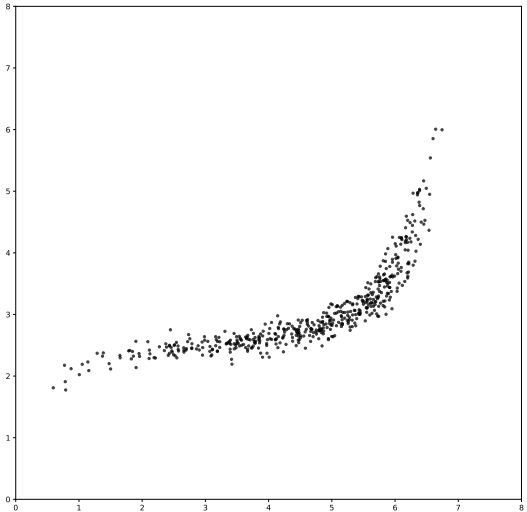
- Scale does not matter $\therefore$ assume keep $= \hat{d} \cdot x$, $|\hat{d}| = 1$

- What is the optimal $\hat{d}$?

UNIVERSITY OF BATH

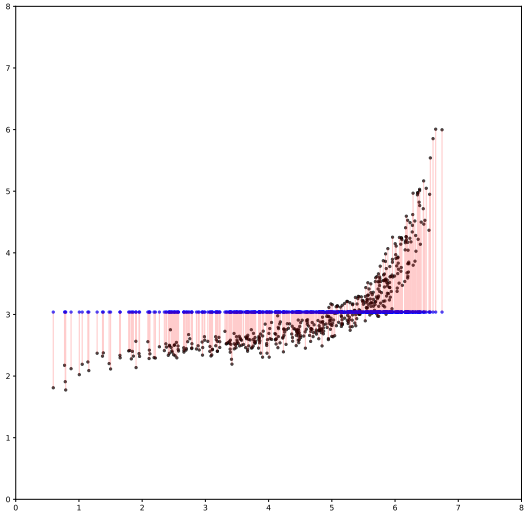
# Visualise I

- Simplest case: 2D to 1D



# Visualise I

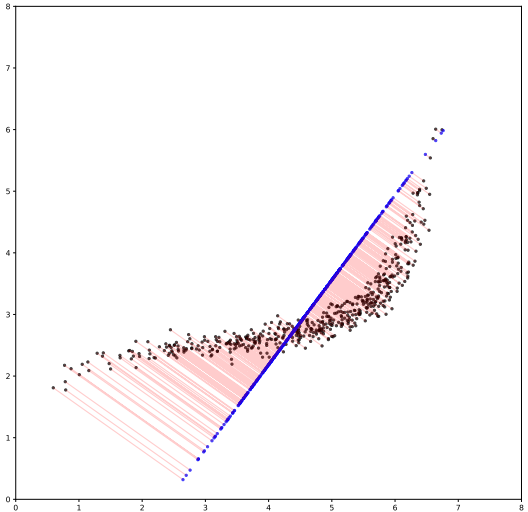
- Simplest case: 2D to 1D
- Linear combination = project onto a line



# Visualise I

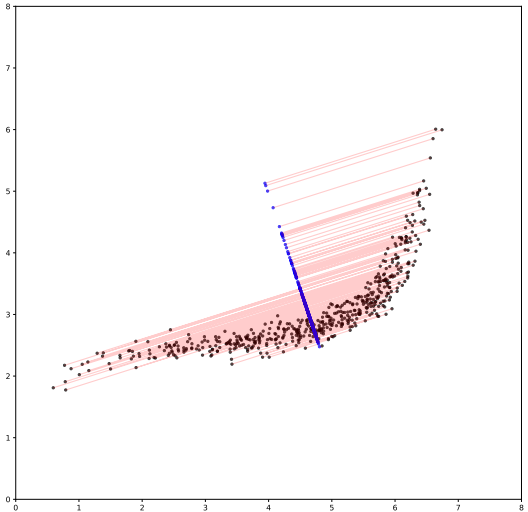
- Simplest case: 2D to 1D
- Linear combination = project onto a line

- Projection distances = information lost
  $\therefore$ minimise! (squared)



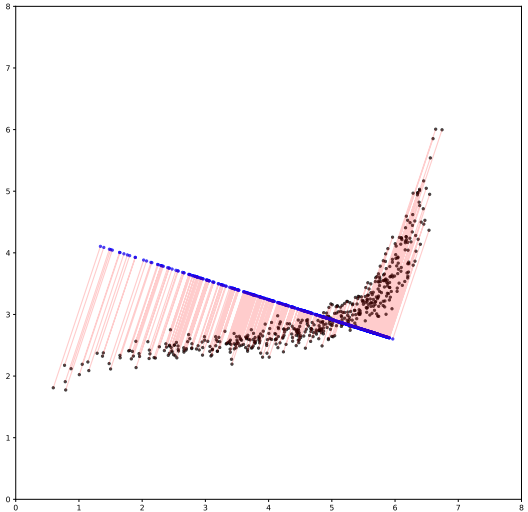
# Visualise I

- Simplest case: 2D to 1D
- Linear combination = project onto a line
- Projection distances = information lost
  $\therefore$ minimise! (squared)
- Line $\therefore$ goes through mean



# Visualise I

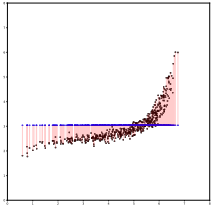
- Simplest case: 2D to 1D
- Linear combination = project onto a line
- Projection distances = information lost
  $\therefore$ minimise! (squared)
- Line $\therefore$ goes through mean
- Best orientation?



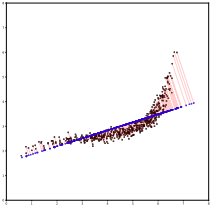
UNIVERSITY OF BATH

# Visualise II

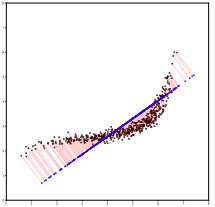
0°:



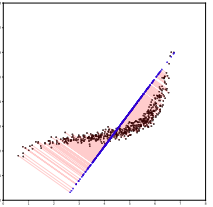
18°:



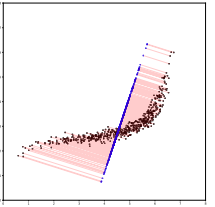
36°:



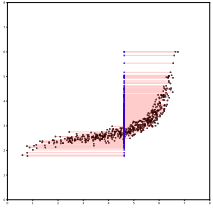
54°:



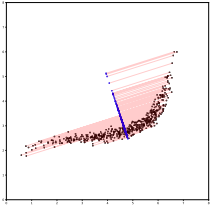
72°:



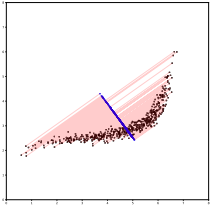
90°:



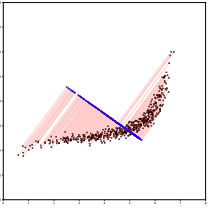
108°:



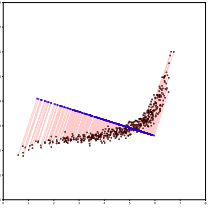
126°:



144°:

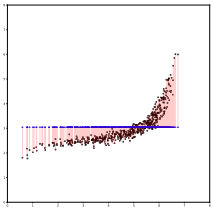


162°:



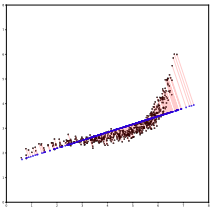
# Visualise II

0°:



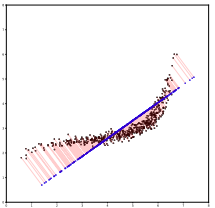
$\sigma^2 = 1.80$

18°:



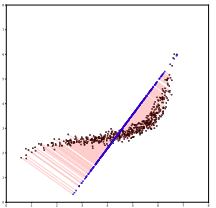
$\sigma^2 = 2.09$

36°:



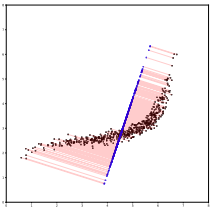
$\sigma^2 = 2.02$

54°:



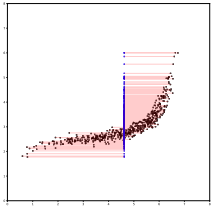
$\sigma^2 = 1.60$

72°:



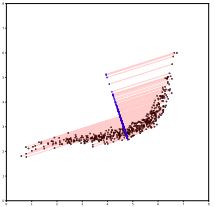
$\sigma^2 = 1.00$

90°:



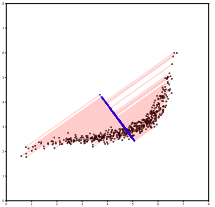
$\sigma^2 = 0.45$

108°:



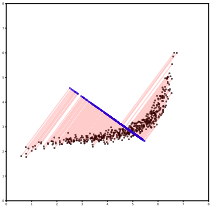
$\sigma^2 = 0.16$

126°:



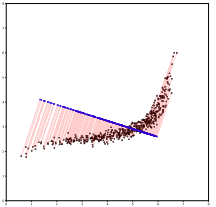
$\sigma^2 = 0.23$

144°:



$\sigma^2 = 0.65$

162°:



$\sigma^2 = 1.25$

# More principle components

- First principle component = direction with **maximum variance**

# More principle components

- First principle component = direction with **maximum variance**
- Second principle component:
  - Select another direction, maximising variance
  - **Orthogonal to first principle component**
  - Orthogonality = no shared (wasted) information

UNIVERSITY OF BATH

# More principle components

- First principle component = direction with **maximum variance**
- Second principle component:
  - Select another direction, maximising variance
  - **Orthogonal to first principle component**
  - Orthogonality = no shared (wasted) information
- Repeat, orthogonal to all previous components each time

  (set of basis vectors – a rotation by convention)

(not actually coded like this)

# Energy

- Energy $= \sum_{i=1}^{n} \sigma_i^2$
  (sum of variance for all features)
- Invariant to rotation!

# Energy

- Energy $= \sum_{i=1}^{n} \sigma_i^2$
  (sum of variance for all features)
- Invariant to rotation!

- PCA is a rotation (followed by throwing away low variance dimensions)
- Maximises variance for each principle component

# Energy

- Energy $= \sum_{i=1}^{n} \sigma_i^2$
  (sum of variance for all features)
- Invariant to rotation!

- PCA is a rotation (followed by throwing away low variance dimensions)
- Maximises variance for each principle component

- PCA minimises lost energy
- Typically: Keep 99.9% of energy
- Select principle components until threshold passed

# Algorithm I

- Has analytic solution!

- The principle components = eigenvectors of covariance matrix
- Variance along eigenvector = eigenvalue

  $\therefore$
- PCA uses a eigendecomposition

# Algorithm II

- Subtract mean
- Eigendecomposition of covariance matrix
- Keep eigenvectors with largest eigenvalues
- Energy = sum of eigenvalues, either
  - Keep 99.9% (for dimensionality reduction)
  - Keep 2 or 3 (for visualisation)
- Multiply data with kept eigenvectors (rotation + deletion of low energy directions)

- Return to original space: (with information loss)
  Multiply with transpose of kept eigenvectors, add back mean

UNIVERSITY OF BATH

# Algorithm III

```
data -= data.mean(axis=0, keepdims=True) # data[exemplar, feature]
covar = numpy.cov(data.T) # Symmetric

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

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

project = evecs[:,start:][:,::-1].T # Create transform
projected = project.dot(data.T).T # Apply to data
```

UNIVERSITY OF BATH

# PCA example

- Data set: Superconductors, 81 features
- Goal: Predict critical temperature

UNIVERSITY OF BATH

# PCA example

- Data set: Superconductors, 81 features
- Goal: Predict critical temperature

- PCA: Reduced to 10 features (99.9% energy)

- Elastic net with 81 features: RMSE = 21.0 kelvin, training time 292 milliseconds
- Elastic net with 10 features: RMSE = 21.0 kelvin, training time 17.5 milliseconds

# Further manifold algorithms

All non-linear: (and all start with PCA, if only for initialisation!)

- t-SNE
- Isomap – Probably most common for visualisation
- Autoencoder – Probably most used for dimensionality reduction
- Probabilistic PCA
- Gaussian process latent variable model – Probably most theoretically sweet

# Other curse-avoiding approaches

- Learning features with convolutional NN
- Using structure (conditional independence)

UNIVERSITY OF BATH

# Classic preprocessing

- Steps:
  1. Whitening

```
data -= data.mean(axis=0, keepdims=True)
data /= data.std(axis=0, keepdims=True)
```

     (subtract mean, divide by standard deviation)

  2. PCA

- Most algorithms: Improves performance (more accurate and faster!)

UNIVERSITY OF BATH

# Summary

- The curse of dimensionality
  (beware your intuitions in high dimensions – they are wrong)
- 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`