COMP6237 Data Mining

Modelling Prices & Nearest Neighbours

Jonathon Hare
jsh2@ecs.soton.ac.uk

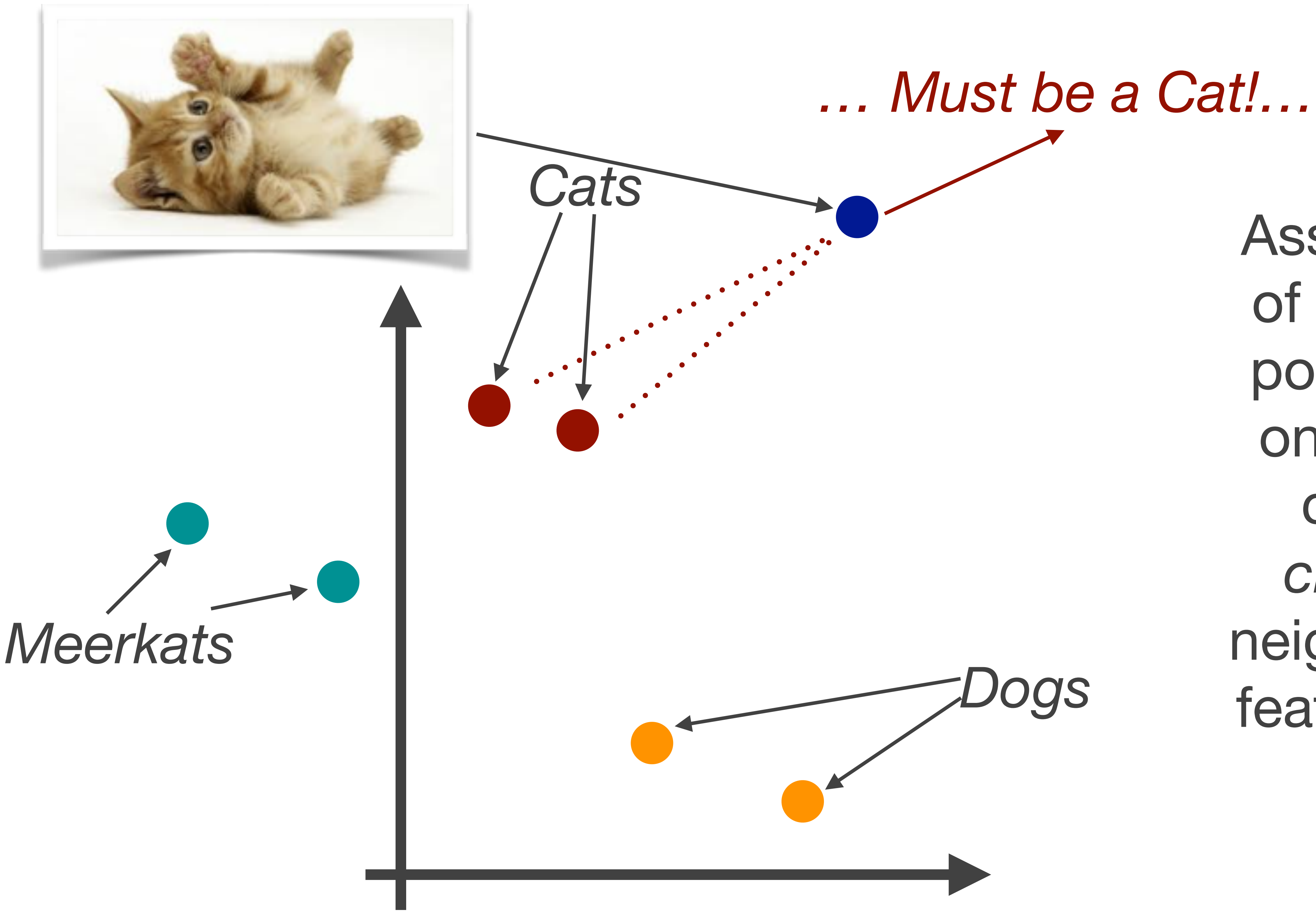
Introduction

- KNN recap
- KNN for regression
- Weighted KNN
- Dealing with KNN limitations
  - Approximate NN
  - Dimensionality reduction again

Problem statement: Price Modelling

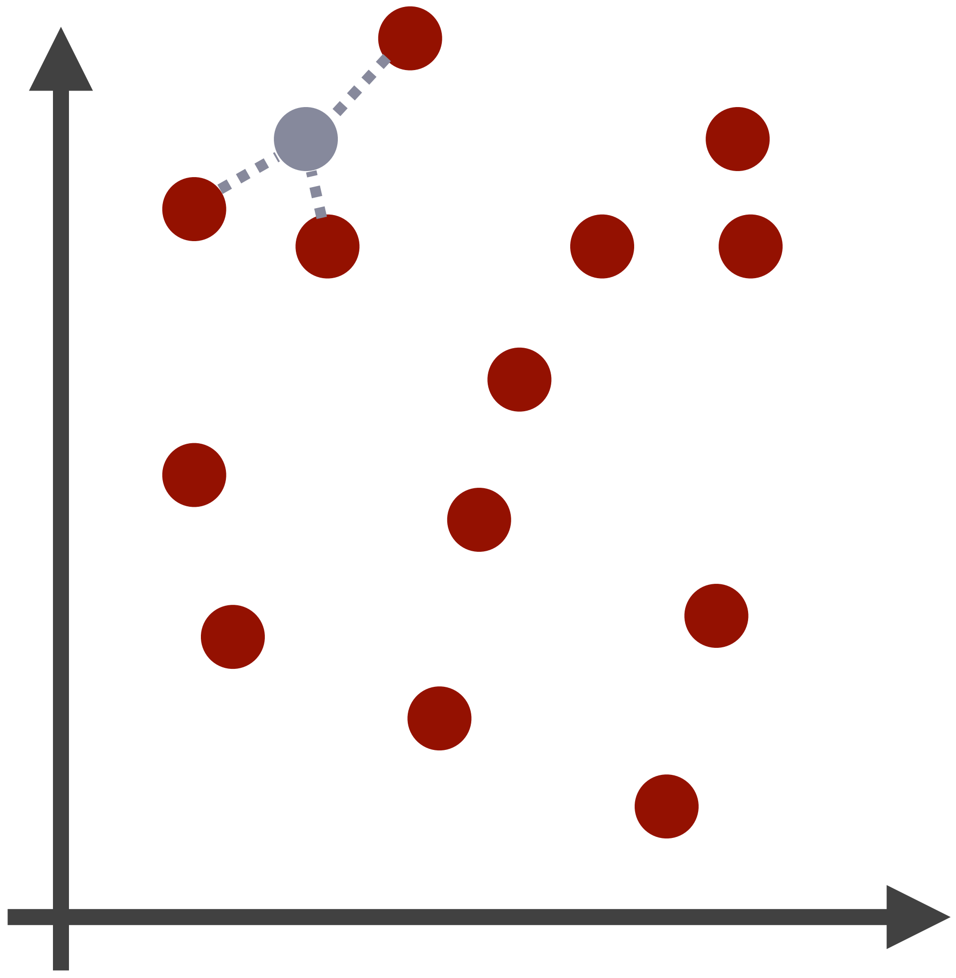
- Assume we want to predict prices of some items
  - Items are described by numerical feature vectors
  - Prices are numbers
  - We have some examples current products and their prices
- Quite clearly a **regression** problem
  - but one with a **potentially complex** regression function

KNN Classification Recap



Assign class of unknown point based on majority class of *closest K* neighbours in featurespace

Using KNN to predict prices (perform regression)



Assign value of unknown point based on *average* values of *closest K* neighbours in featurespace

Example Dataset: Fine wine prices

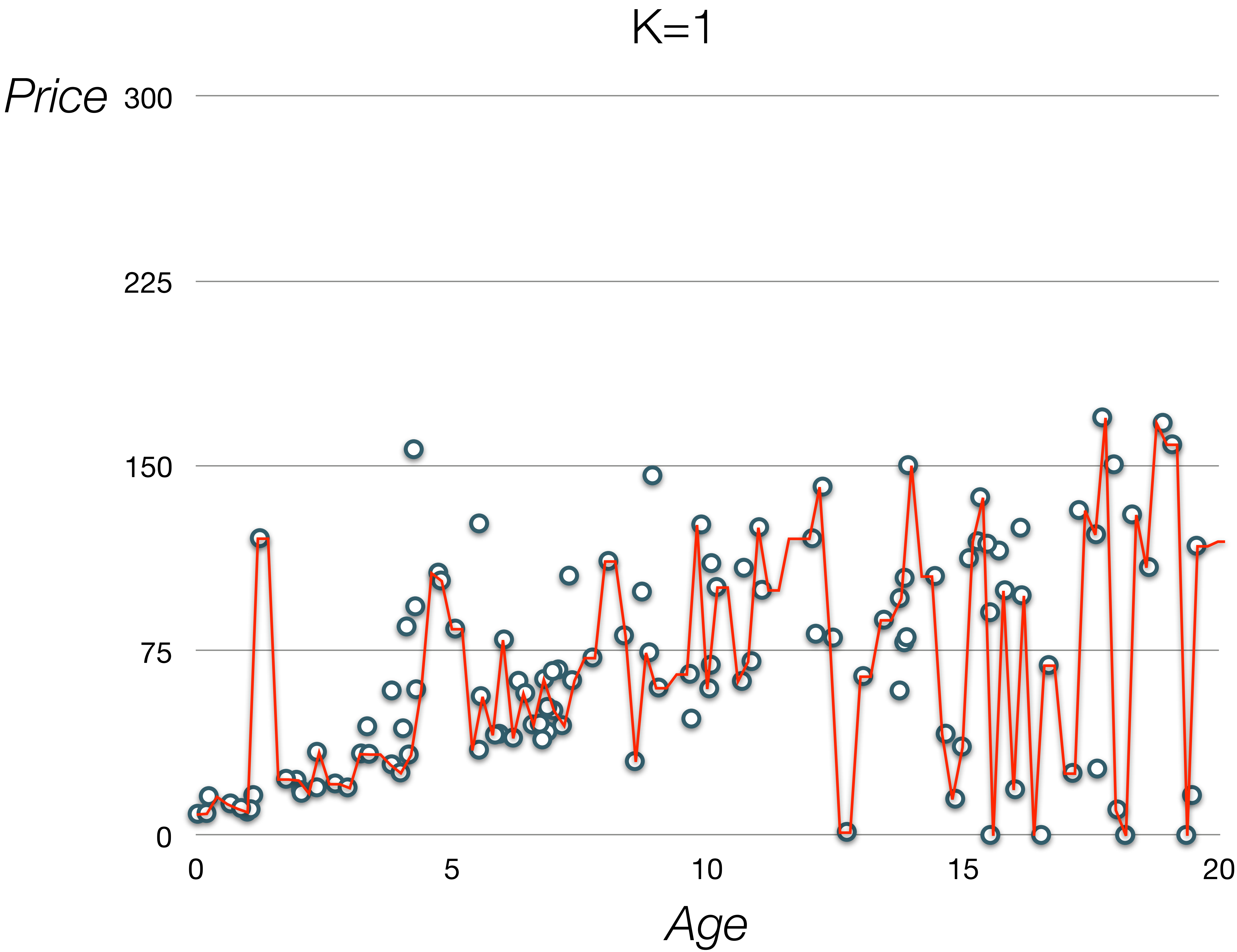
rating %	age	price
73.58	21.92	£188.87
71.87	17.28	£164.59
60.35	31.72	£0.00
71.21	1.65	£23.72
55.92	19.20	£0.00
62.75	27.91	£0.00
71.60	12.69	£108.44
67.25	11.99	£121.98
82.64	43.89	£0.00
97.70	1.05	£9.94
53.78	27.09	£0.00
...	...	...

Demo KNN Regression

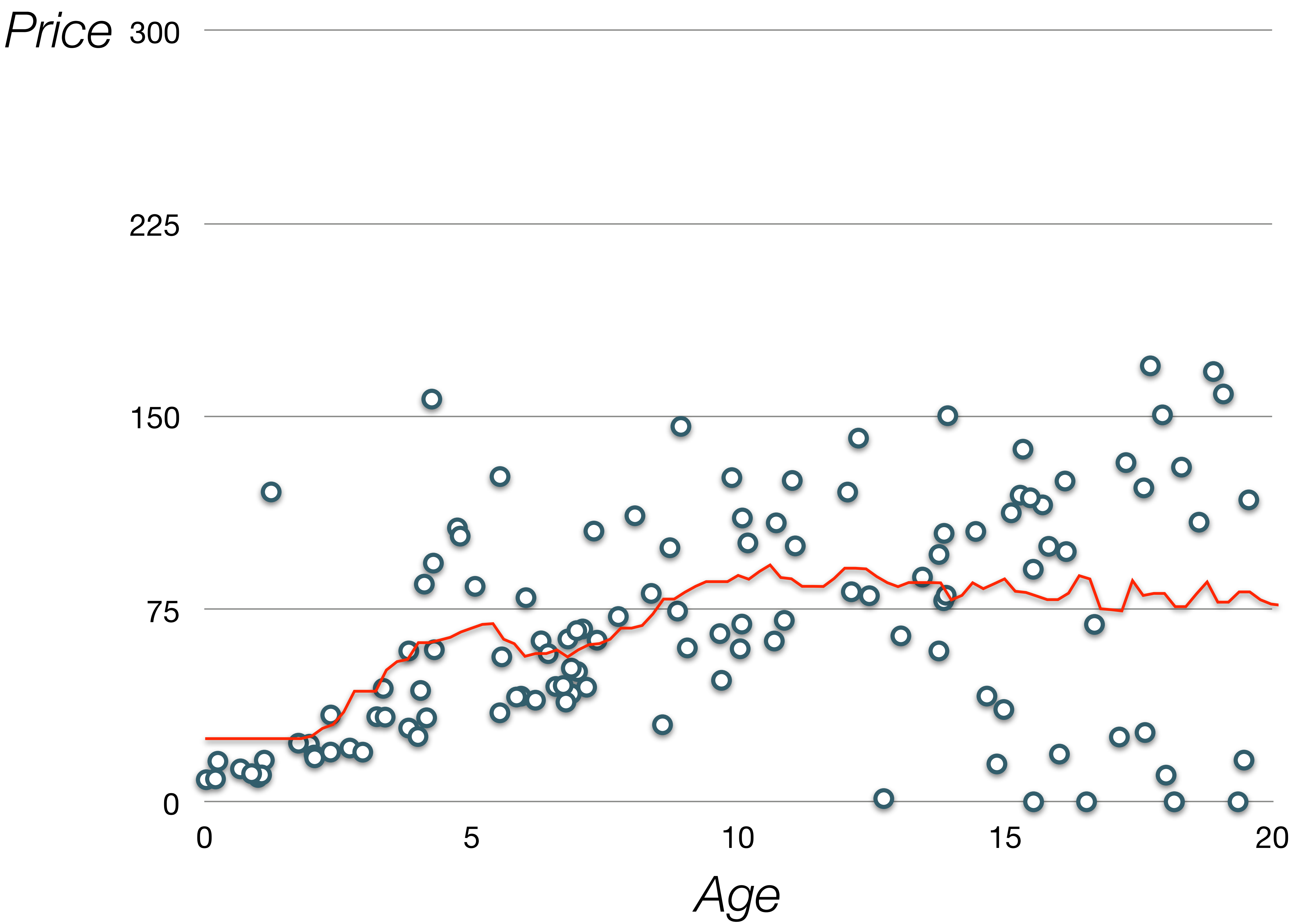
Choosing the correct K

- …can be very important!

- Too small and we might overfit to any noise
  - For regression this can be particularly important
- Too big and we smooth too much…


K=1
Price
300
225
150
75
0
0
5
10
15
20
Age

K=20



Choosing the correct K

- Measure performance (e.g. regression or classification error) using cross validation

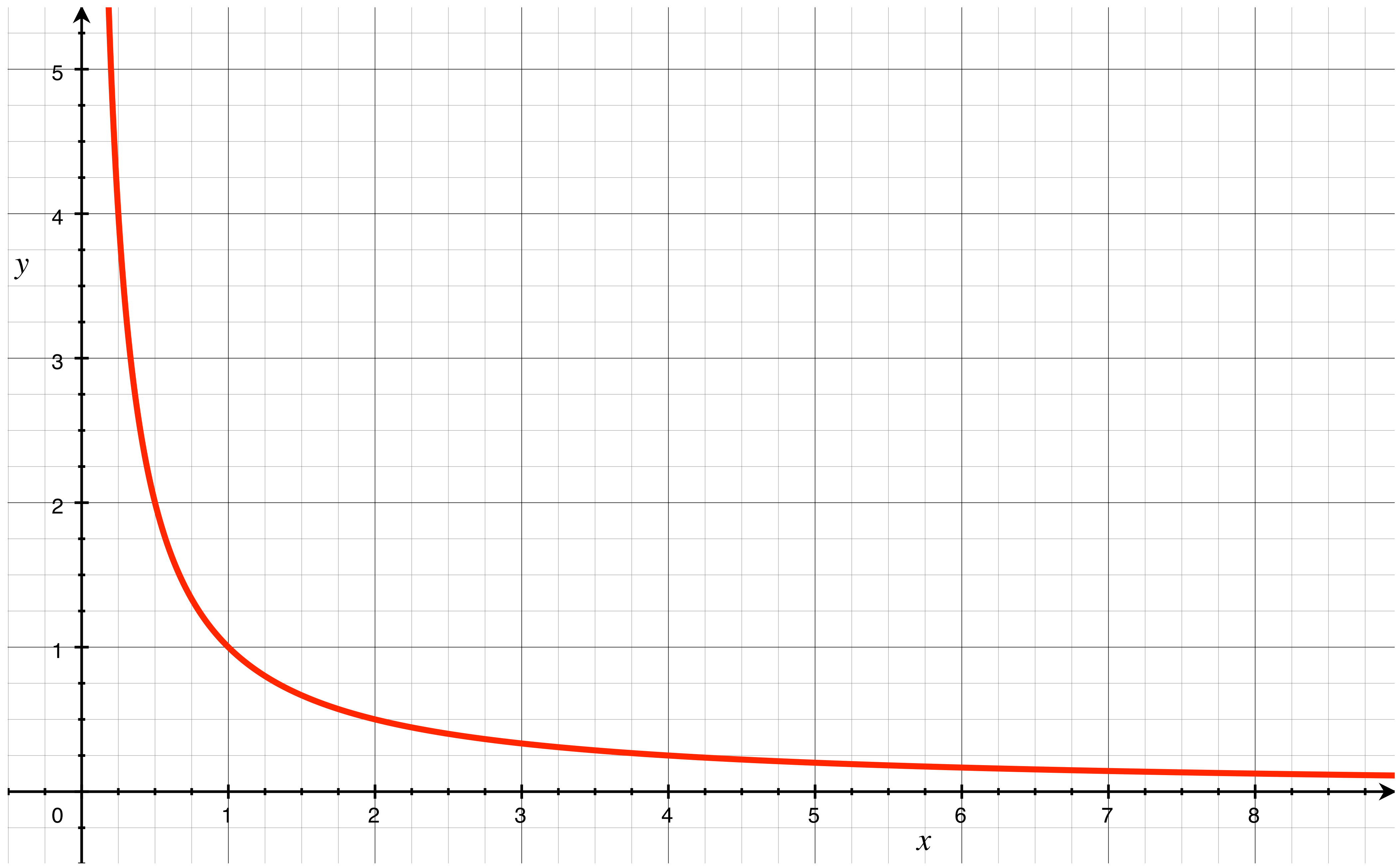
- Optimise k to maximise performance

Improving KNN: Weighting Neighbours

- Rather than taking the average value of the neighbours (or mode for classification), weight neighbours based on their distance
    - Intuition: neighbours further away are likely to be less similar, so should have less effect

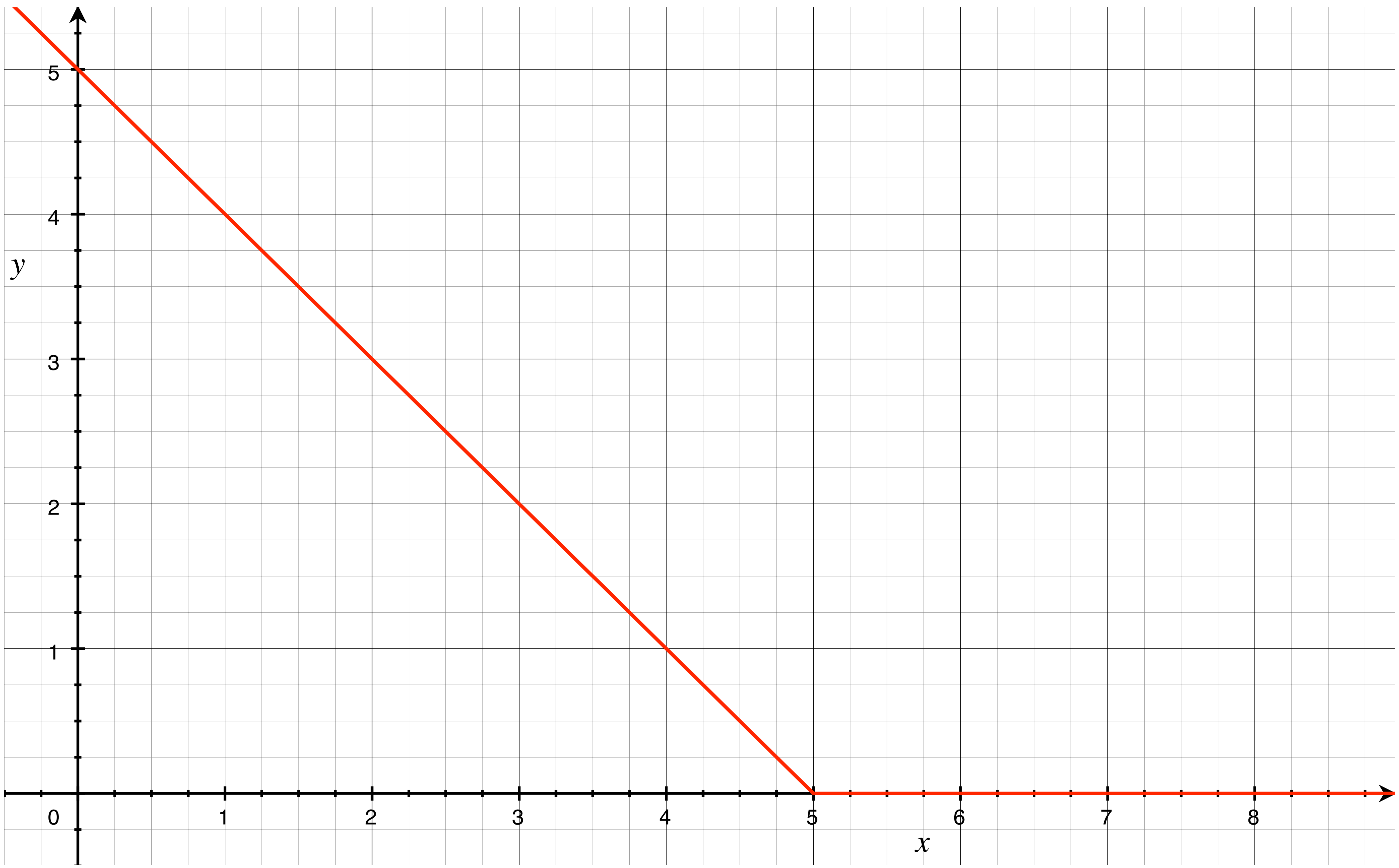
Inverse weighting

weight=k_1/(distance + k_2)



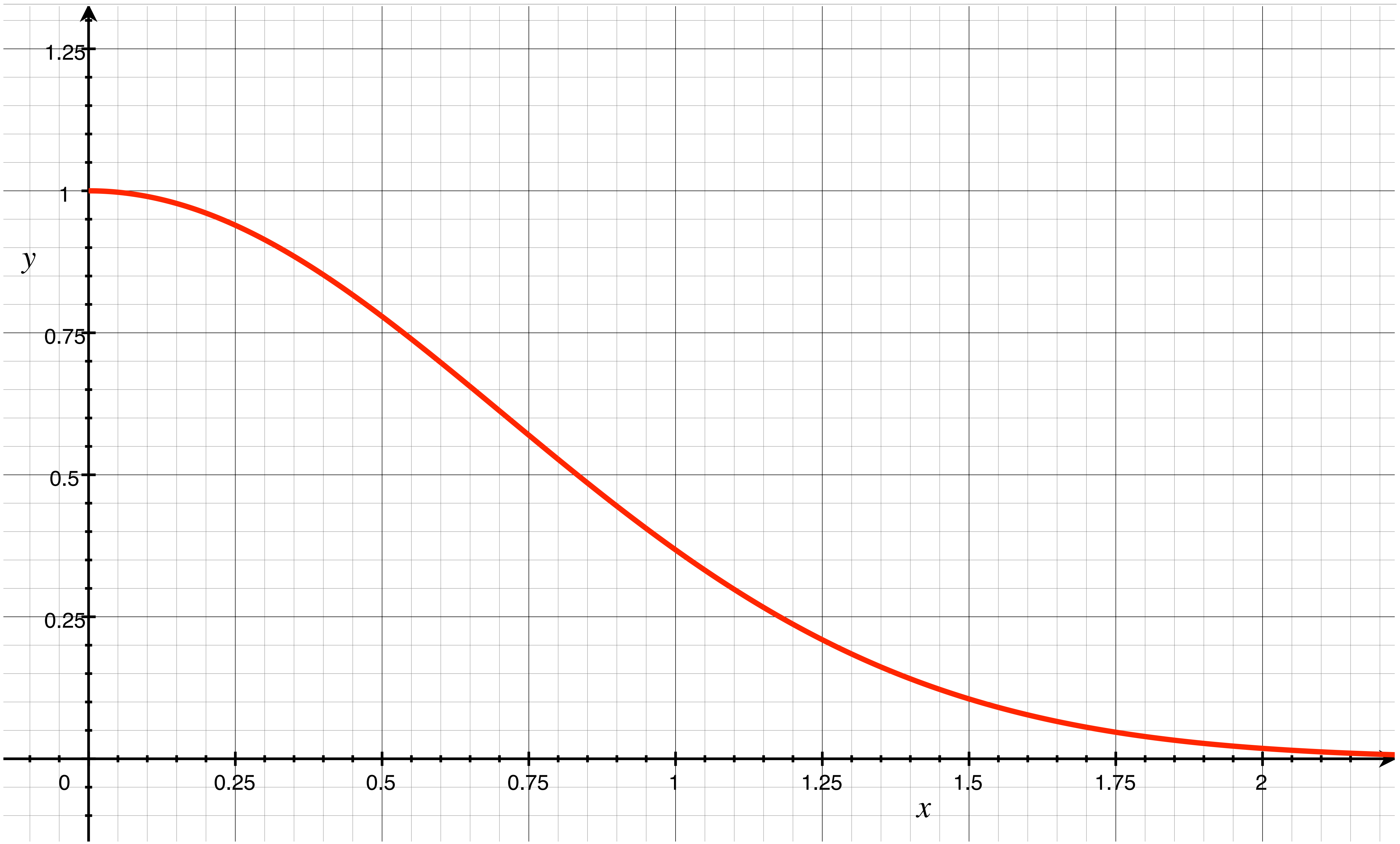
Subtraction weighting

weight=max(0,k1-distance)



Gaussian weighting

weight=exp(-distance^2/k_1^{2})



Demo Weighted KNN Regression

Choosing the best weighting

- Measure performance (e.g. regression or classification error) using cross validation
- Optimise weighting scheme & parameters to maximise performance

Dealing with Heterogeneous Variables

- In most datasets the variables/features don’t have the same range or scale of values
    - But if we’re computing distances this is obviously important - features with bigger ranges would have more of an impact
- What about entirely irrelevant variables?
    - Might force things to be far apart even though they should be considered to be similar…

Example Dataset: Fine wine prices v2

rating %	age	aisle	bottle size	price
70.37	38.89	8	750	£0.00
77.48	39.67	9	375	£0.00
90.01	45.30	18	375	£51.25
80.16	11.76	8	750	£86.45
94.56	1.00	19	750	£176.96
67.75	30.78	10	375	£0.00
97.82	43.95	1	750	£244.00
81.41	6.77	2	750	£0.00
96.40	36.48	17	375	£100.85
82.73	48.58	14	375	£0.00
75.14	1.26	14	375	£8.83
52.15	18.51	19	750	£0.00

Normalising by scaling dimensions

- Normalisation can help
  - e.g. bring all features to same range
  - *but is this optimal?*
    - might it be better to choose scale factors for each feature that together optimise the performance?
      - Could also use this to perform *feature selection*

Demo: Optimising feature weightings

KNN Problems

- **Computationally expensive** if there are:
  - Lots of training examples
  - Many dimensions
- However:
  - More examples generally means better accuracy
  - More dimensions generally gives more descriptive power (unless dimensions are highly correlated or irrelevant…)

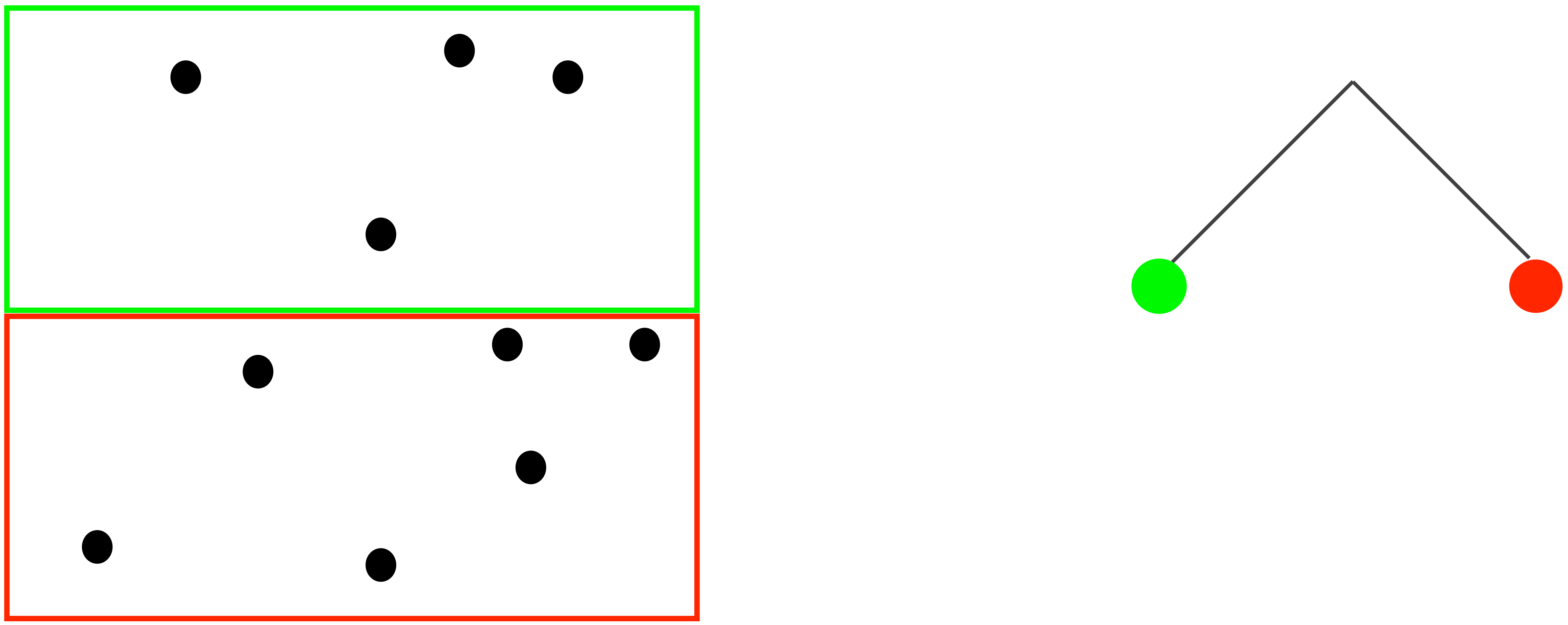
Dealing with lots of examples: Approximate NN

- How can we **quickly** find the nearest neighbour to a query point in a high dimensional space?
  - Index the points in some kind of tree structure?
  - Hash the points?
  - Quantise the space

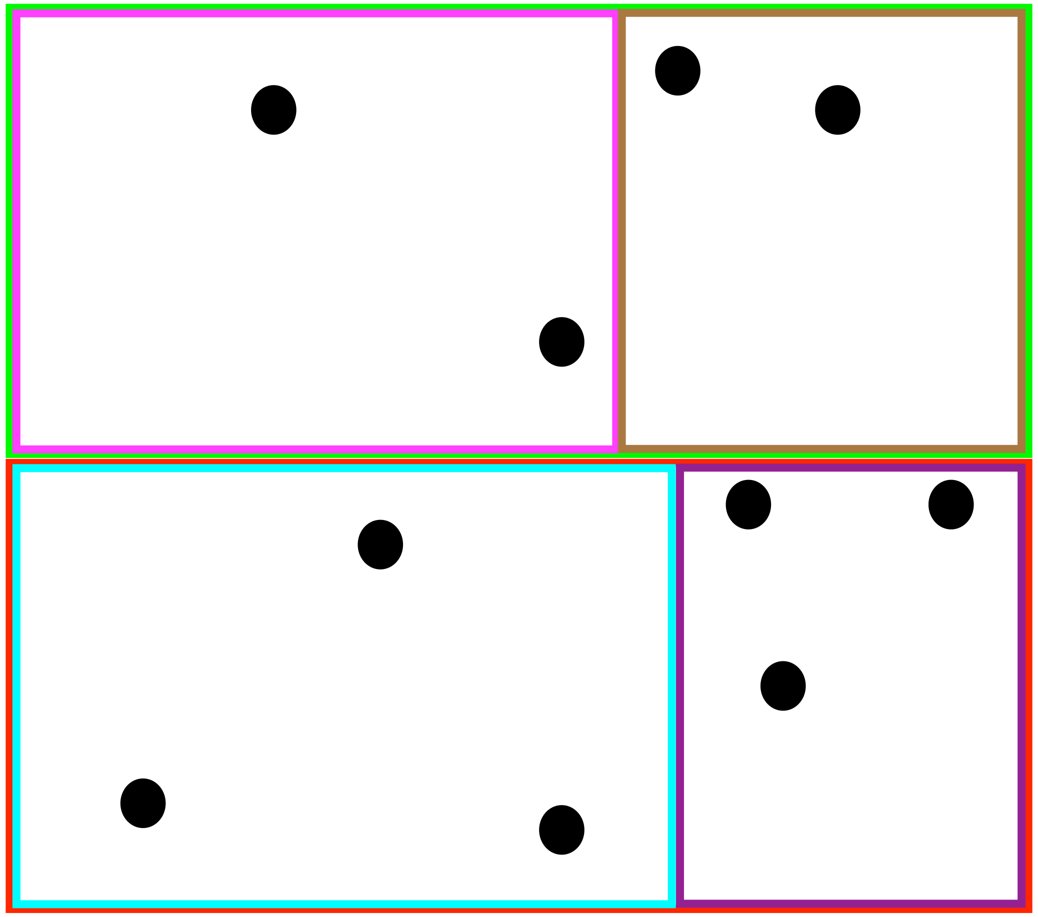
K-D Trees

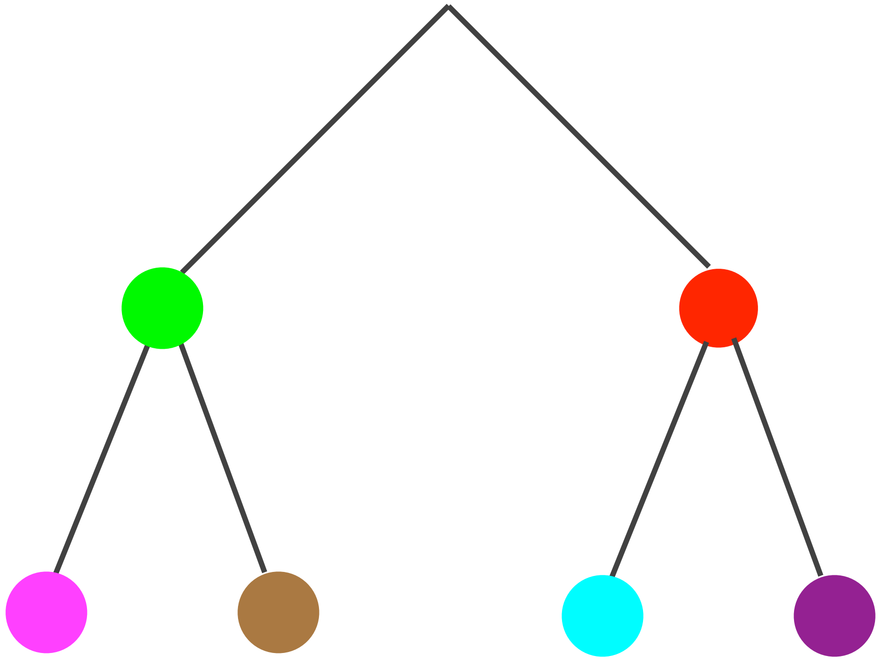
- Binary tree structure that partitions the space along axis-aligned hyperplanes
  - Typically take each dimension in turn and splits on the median of the points in the enclosing partition.
  - Stop after a certain depth, or when the number of points in a leaf is less than a threshold

K-D Trees



K-D Trees

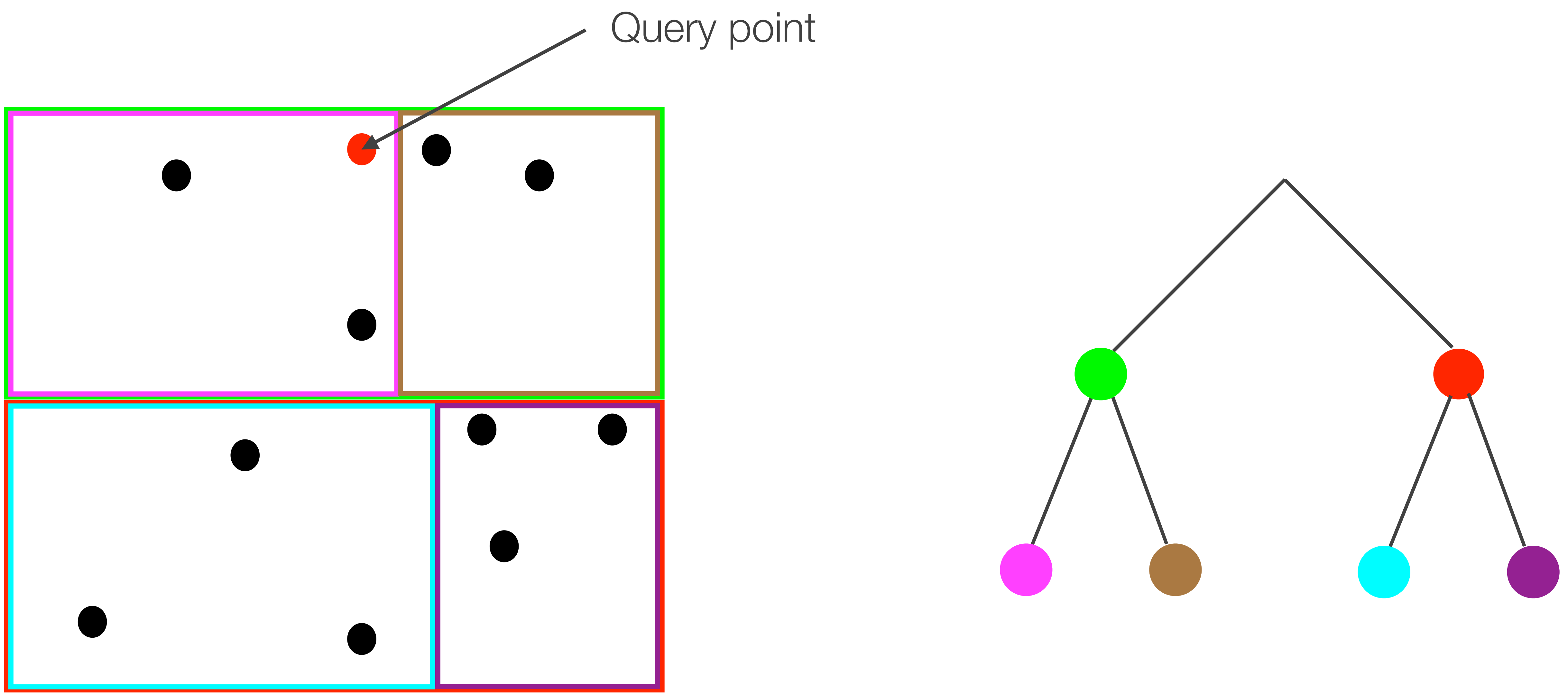




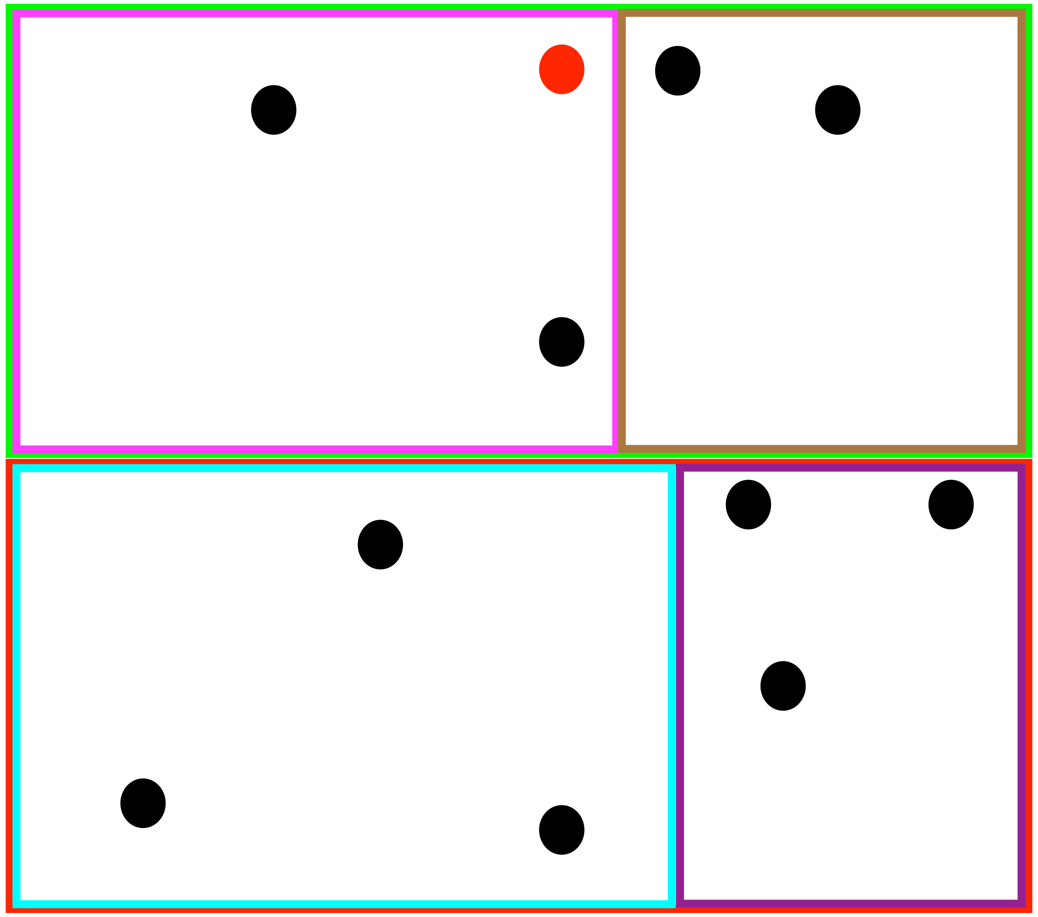
K-D Trees

- Search by walking down the tree until a leaf is hit, and then brute-force search to find the best in the leaf.
  - This is not guaranteed to be the best though…
  - To have to walk back up the tree and see if there are any better matches, and only stop once the root is reached (note you don’t have to check a subtree if it’s clear that all points in that subtree are further than the current best).

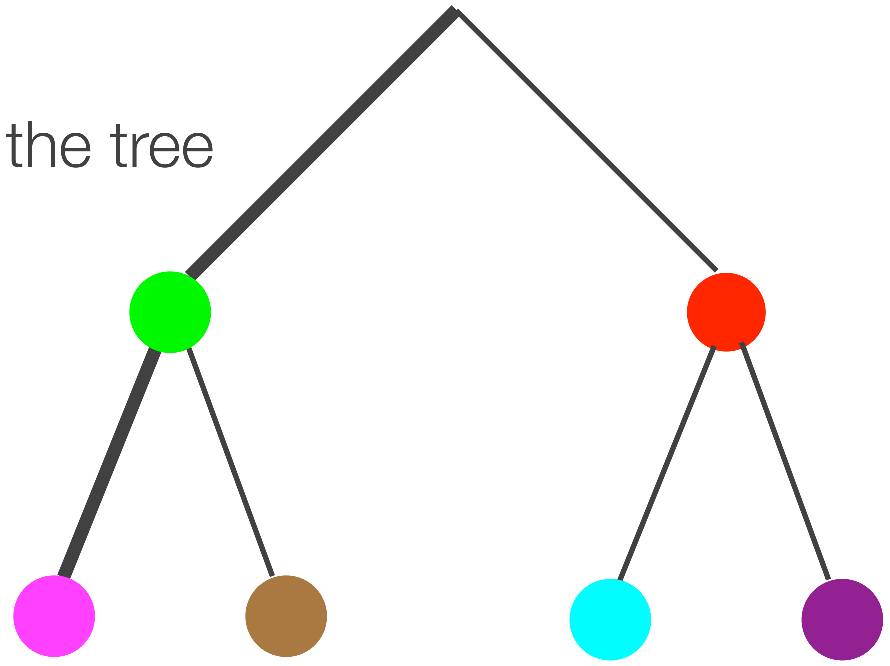
K-D Trees



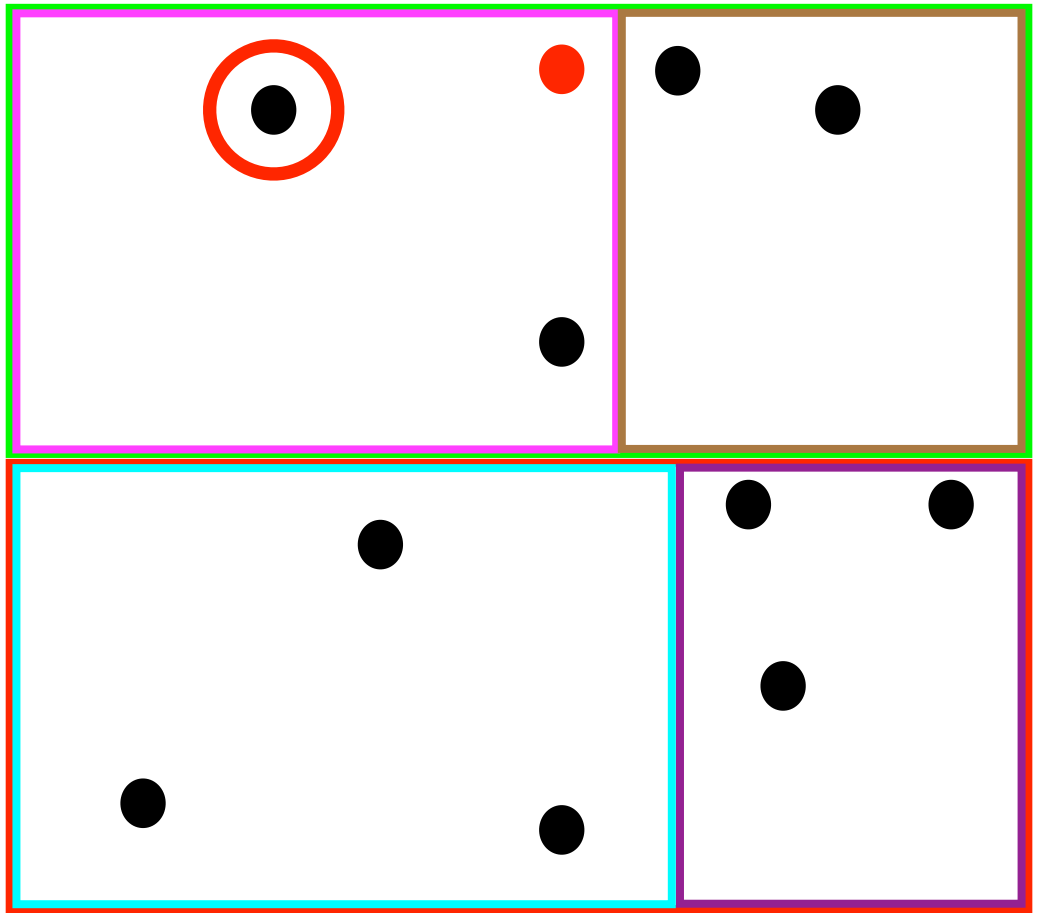
K-D Trees



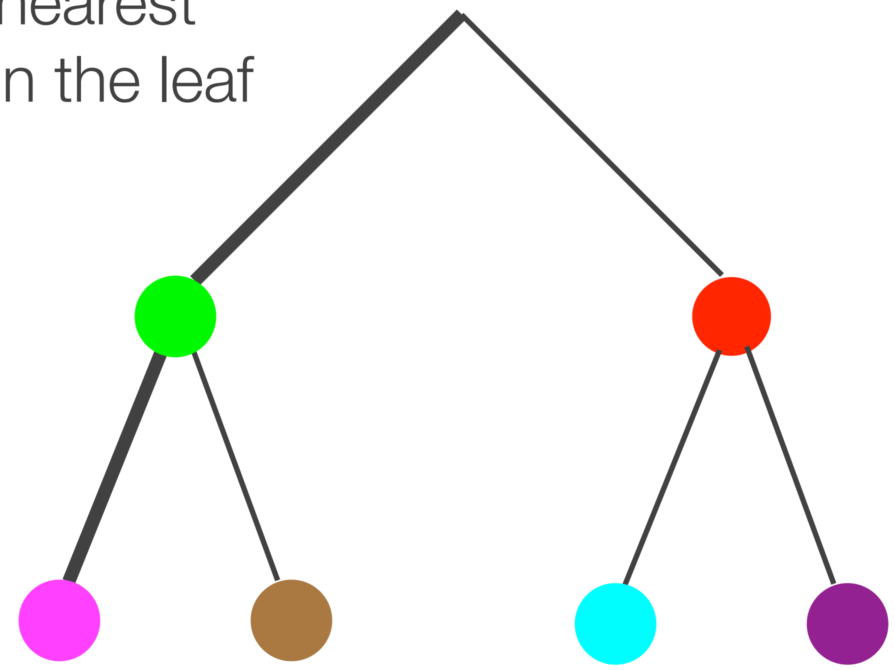
Walk down the tree



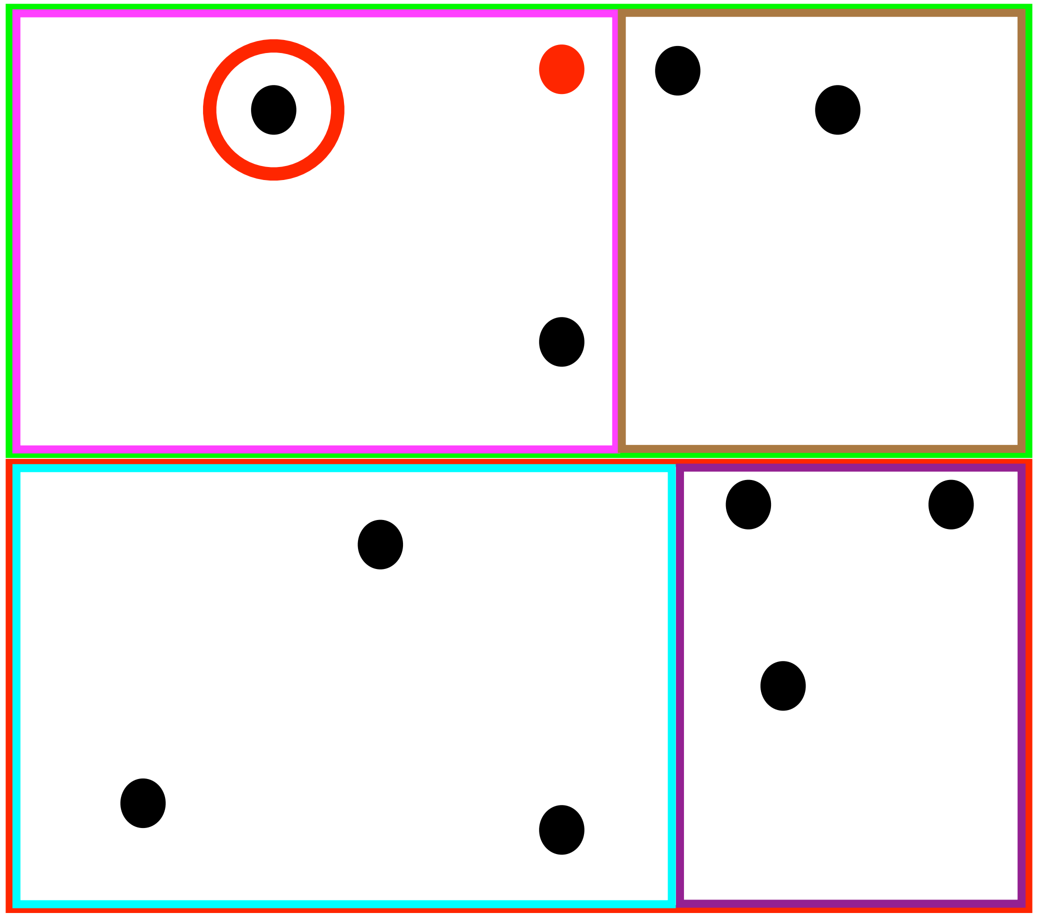
K-D Trees



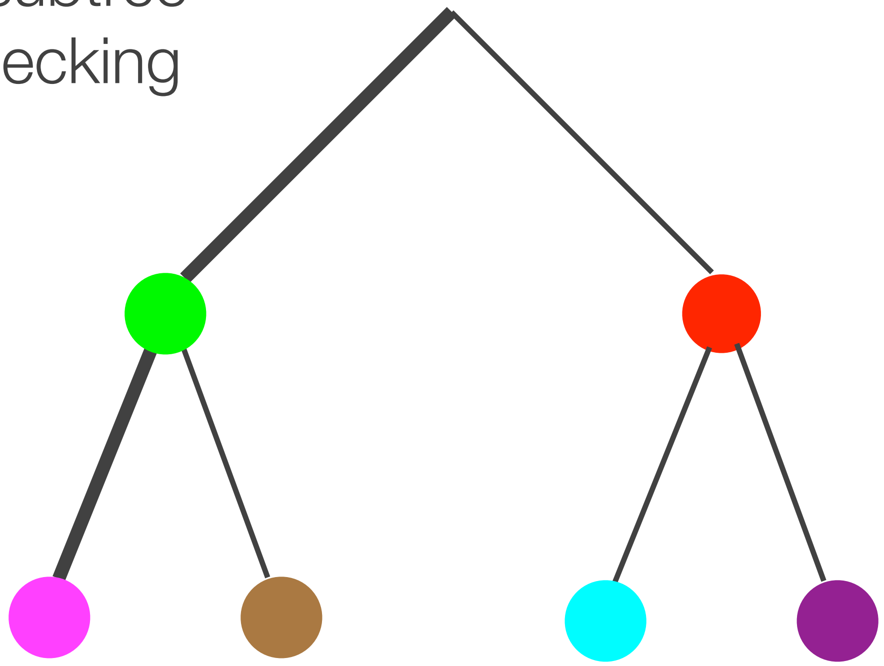
Find the nearest neighbour in the leaf



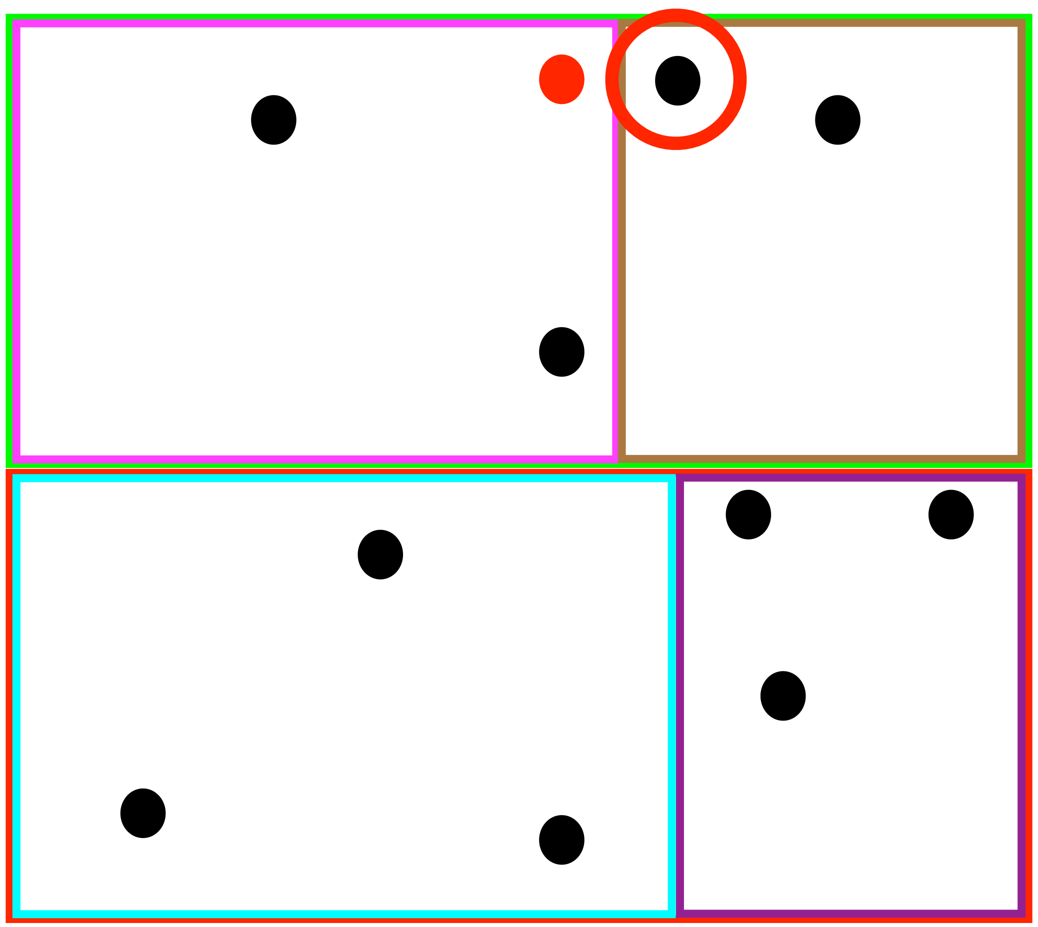
K-D Trees



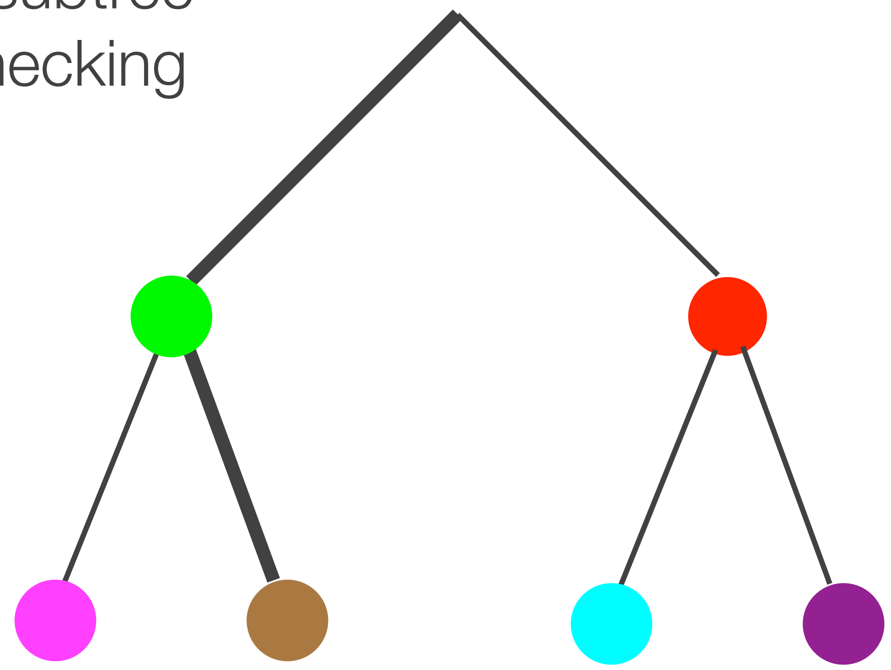
Backtrack, and see if
the next subtree
needs checking



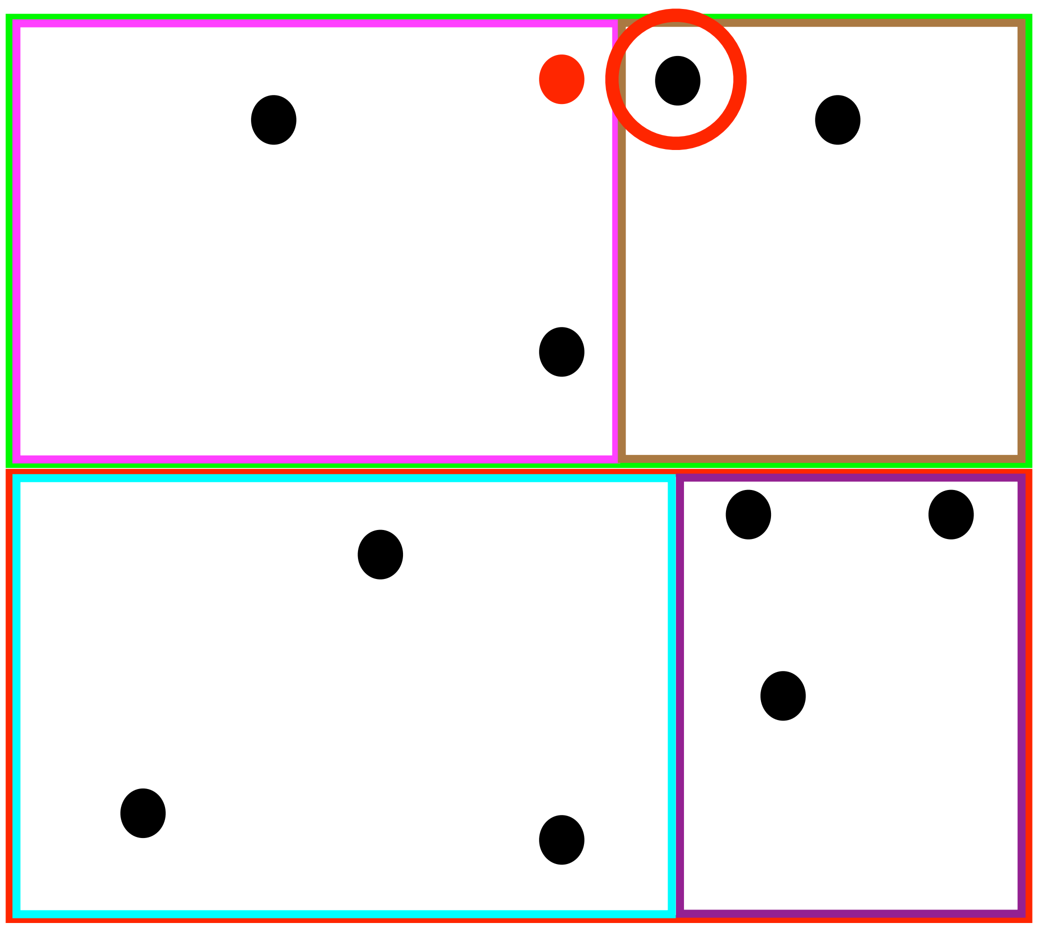
K-D Trees

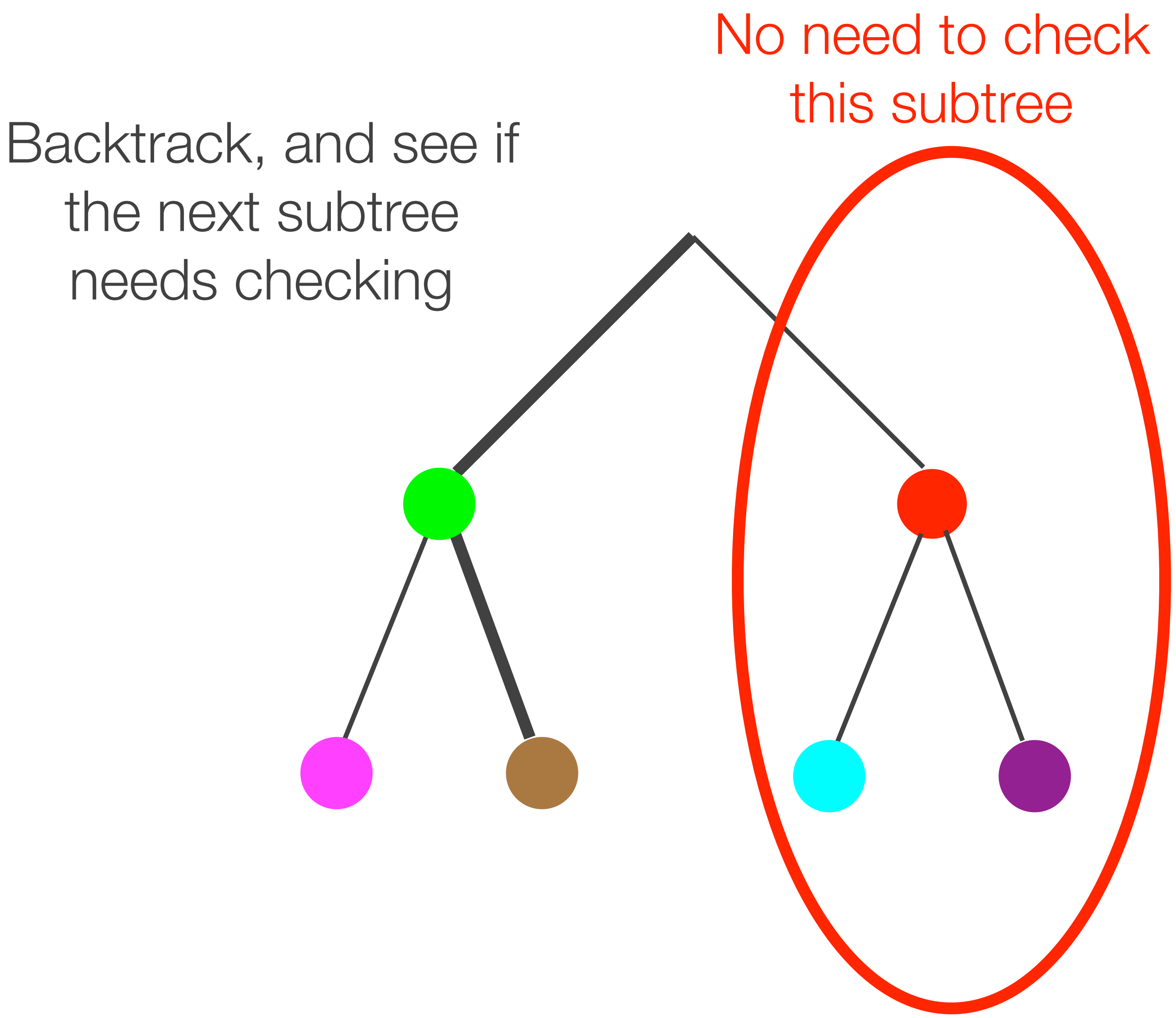


Backtrack, and see if
the next subtree
needs checking



K-D Trees





No need to check
this subtree
Backtrack, and see if
the next subtree
needs checking

K-D Tree problems

- Doesn't scale well to high dimensions
  - You tend to end up needing to search most of the tree
- There are *approximate* versions that won't necessarily return the exact answer that do scale (if you don't mind the potential for mismatch)
  - typically based on ensembles of trees where the split dimension is randomised (but biased towards the *best*)

Locality Sensitive Hashing

- Locality Sensitive Hashing (LSH) creates hash codes for vectors such that similar vectors have similar hash codes!
  - Input vectors hashed so that similar items map to the same "**buckets**" with **high probability**
    - number of buckets is much smaller than the set of input items
  - LSH differs from *conventional* and *cryptographic* hash functions because it aims to **maximise** the **probability** of a "**collision**" for similar items

Metric Space

- $\mathcal{M}=(M,d)$
- M is a set (of vectors)
- d is a distance metric (e.g. a function $d : M \times M \longrightarrow \mathbb{R}$)
    - such that for any x, y, $z \in M$
        - $d(x, y) \geq 0 \Leftrightarrow x=y$
        - $d(x, y) = 0$
        - $d(x, y) = d(y, x)$
        - $d(x, z) \leq d(x, y) + d(y, z)$

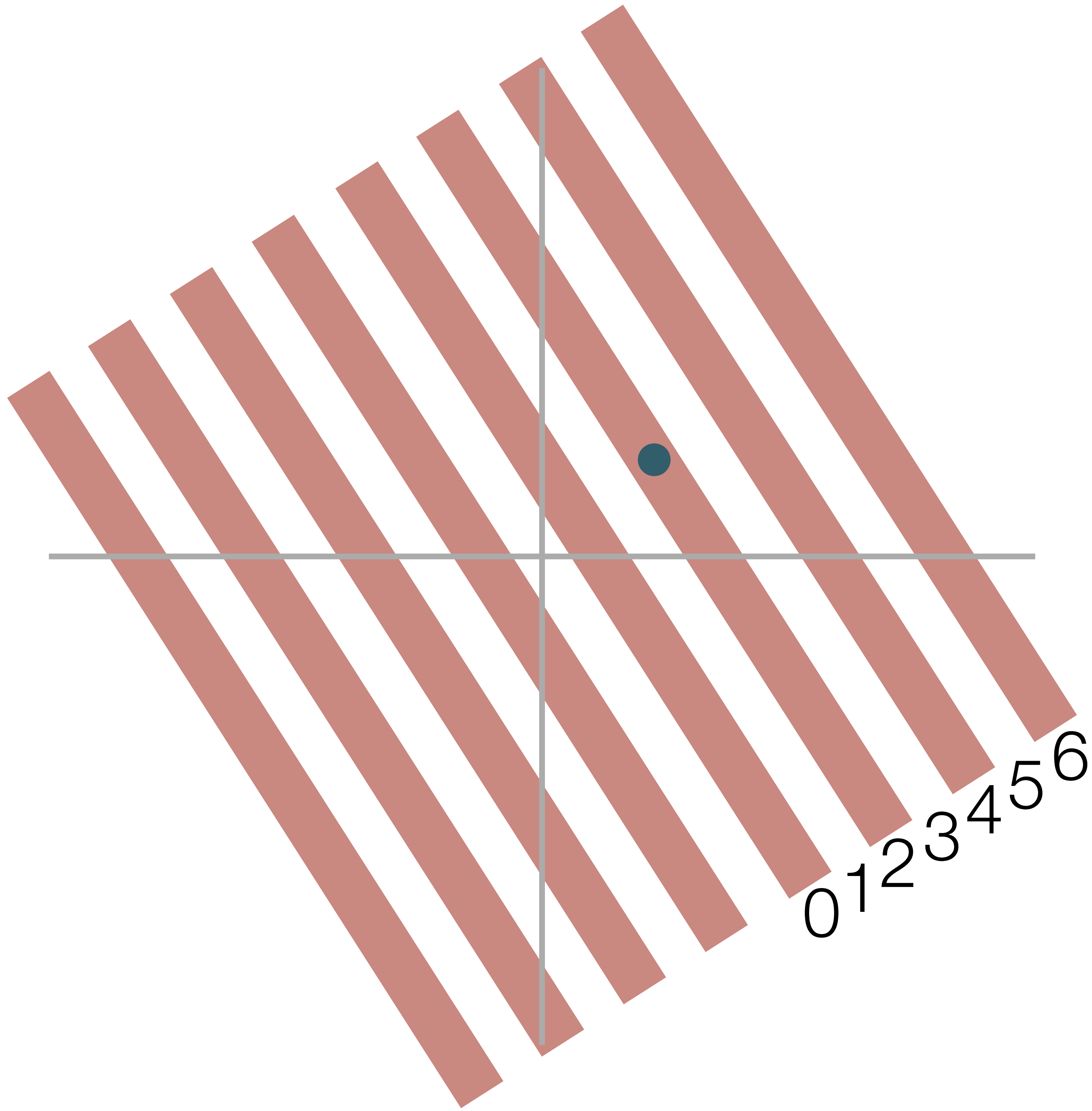
LSH Family

- An LSH family is defined for a
  - metric space $\mathcal{M}=(M,d)$
  - a threshold $R>0$
  - and an approximation factor $c>1$
- The family $\mathcal{F}$ is a family of functions $h : \mathcal{M} \longrightarrow S$ which map elements from the metric space to a bucket $s \in S$.
- $\mathcal{F}$ satisfies the following conditions for any two points p, $q \in M$, using a function $h \in \mathcal{F}$ which is chosen uniformly at random:
  - if $d(p, q) \leq R$, then $h(p) = h(q)$ (i.e. p and q collide) with probability at least P_1,
  - if $d(p, q) \geq cR$, then $h(p) = h(q)$ with probability at most P_2.
- A family is **interesting** when $P_1 > P_2$

Stable Distribution LSH Family

- $h_{\mathbf{a},b}(\mathbf{v}) = \text{floor}((\mathbf{a} \cdot \mathbf{v} + b)/r)$
  - ***a*** is a *d* dimensional vector with elements independently randomly generated from a *stable* distribution
  - *b* is a uniform random number in [0,*r*]
- Interesting factoids:
  - if the elements of ***a*** are drawn from a Gaussian distribution then underlying metric is Euclidean
  - if the elements of ***a*** are drawn from a Cauchy distribution then underlying metric is L1

Geometric Interpretation



Nearest Neighbour Search with LSH

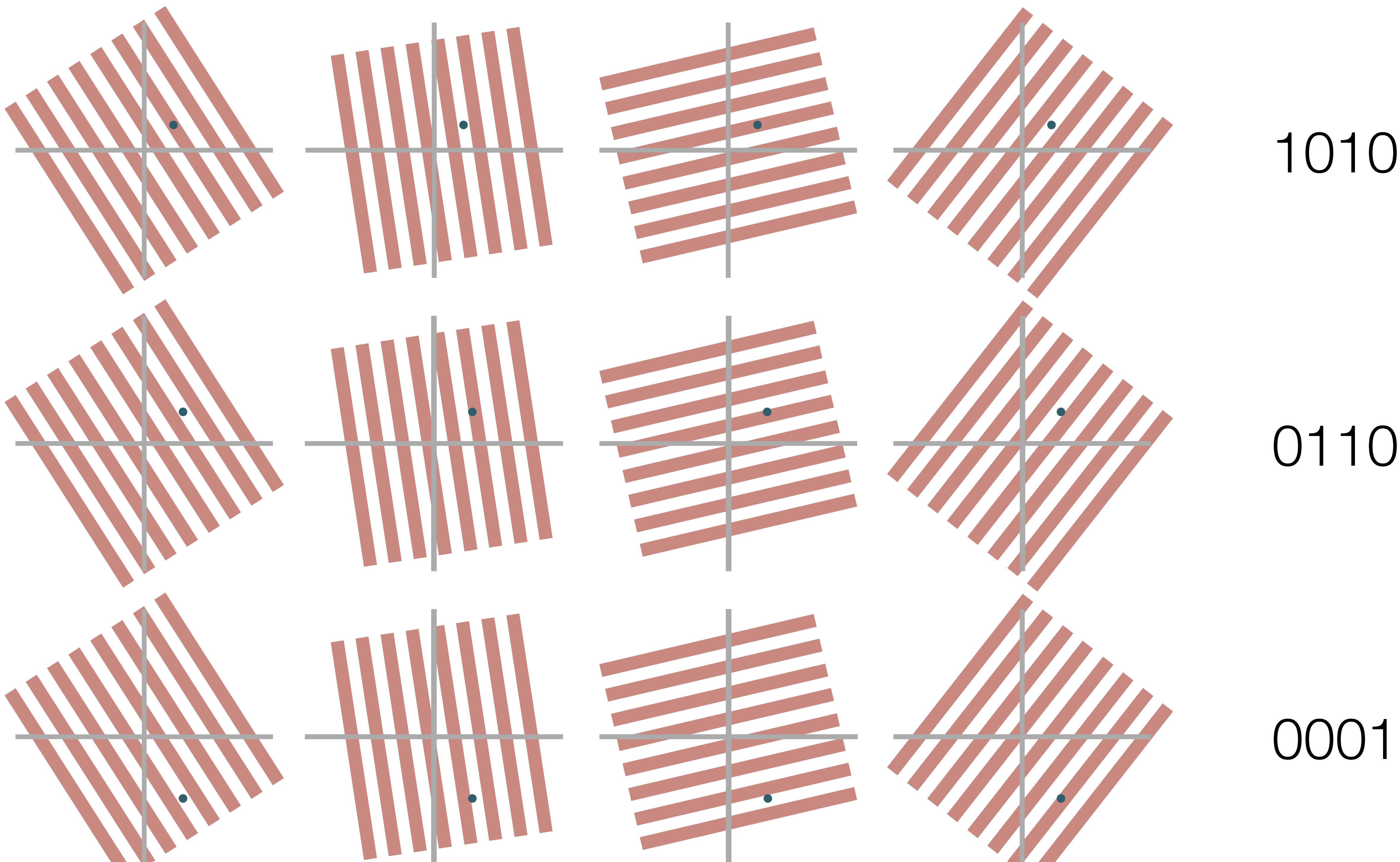
- Can use LSH as a basis for NN search using standard hash tables
  - Combine responses of a list of LSH's into a vector of ints
    - Store in a hash table of <int[], Set of points>
  - Use multiple hash tables to better ensure probability of collision

Sketching

- A technique called sketching concatenates binary hashes into a bit string.
  - With the correct LSH function, the Hamming distance between a pair of sketches is proportional to the Euclidean distance between the original vectors
  - Can easily compress features to relatively small number of bits
    - Hamming distance computation is cheap
      - Lookup tables and bitwise operations

Sketching with a Stable (Gaussian) LSH Family

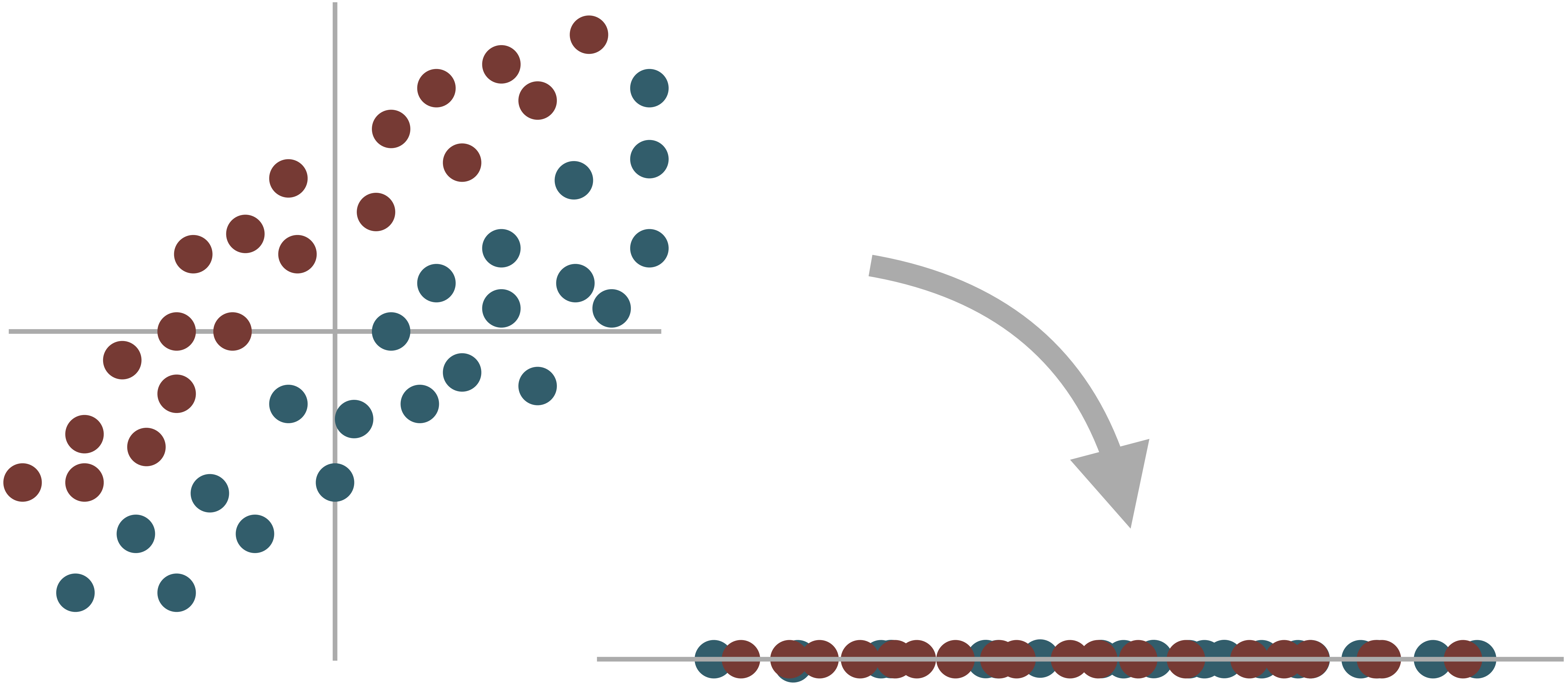
- $h_{\boldsymbol{a},b}(\boldsymbol{v}) = \mathrm{lsb}(\mathrm{floor}((\boldsymbol{a}\cdot\boldsymbol{v}+b)/r))$



Dealing with dimensions: Dimensionality Reduction

- Too many dimensions?
  - Reduce the number…
  - Already looked at a few techniques a few lectures ago
    - Some of those (MDS, SOM) not really suitable for this in terms of doing NN analysis…
    - What about PCA?

PCA has a potential problem…



Random Projection

- Johnson-Lindenstrauss lemma
  - “if points in a vector space are of sufficiently high dimension, then they may be projected into a suitable lower-dimensional space in a way which approximately preserves the distances between the points”
- The low dimensional basis on which we project could be **generated randomly**!

Summary

- KNN can be used for regression as well as classification
- Adding a weighting can improve performance (esp. for regression)
- Some of the problems associated with KNN can be overcome through
  - fast approximate nearest neighbour methods
  - dimensionality reduction