Data Mining
Lecture 9: Nearest Neighbours

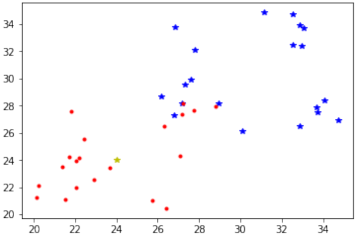
Jo Grundy

ECS Southampton

13^{th} March 2023

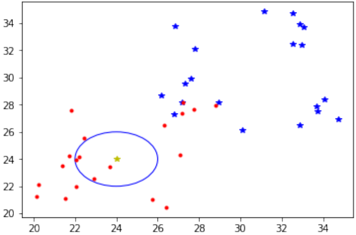
Nearest Neighbours - Introduction

How would you classify this point?



Nearest Neighbours - Introduction

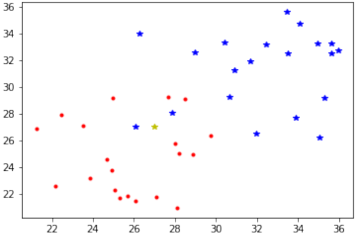
Use the closest samples..



K-Nearest Neighbours: Assigns class based on majority class of closest K neighbours in featurespace

Nearest Neighbours

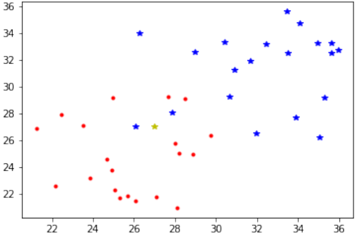
K-Nearest Neighbours: Assigns class based on majority class of closest K neighbours in featurespace



- K = 1?

Nearest Neighbours

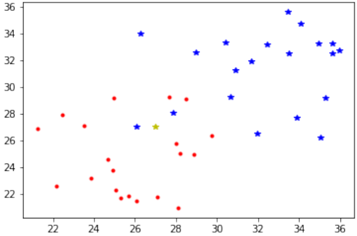
K-Nearest Neighbours: Assigns class based on majority class of closest K neighbours in featurespace



- K = 1? blue star
- K = 3?

Nearest Neighbours

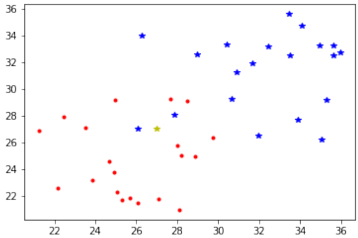
K-Nearest Neighbours: Assigns class based on majority class of closest K neighbours in featurespace



- $K = 1$? blue star
- $K = 3$? blue star
- $K = 5$?

Nearest Neighbours

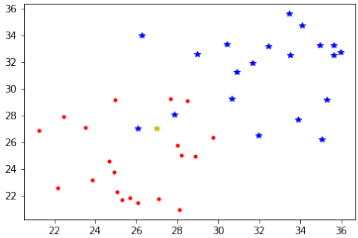
K-Nearest Neighbours: Assigns class based on majority class of closest K neighbours in featurespace



- $K = 1$? blue star
- $K = 3$? blue star
- $K = 5$? red dot
- $K > 5$?

Nearest Neighbours

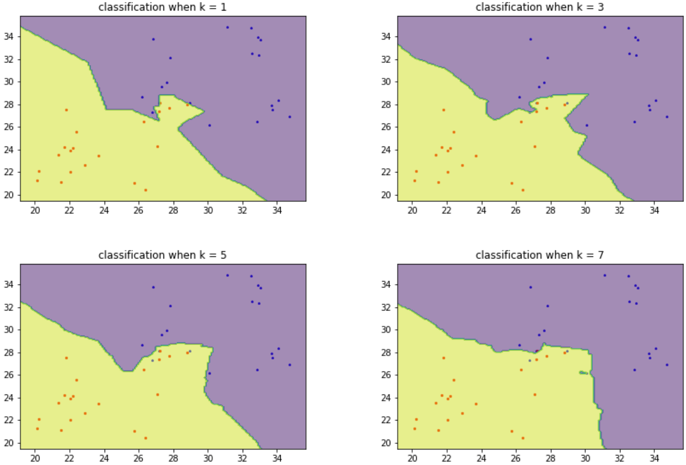
K-Nearest Neighbours: Assigns class based on majority class of closest K neighbours in featurespace



- $K = 1$? blue star
- $K = 3$? blue star
- $K = 5$? red dot
- $K > 5$? red dot

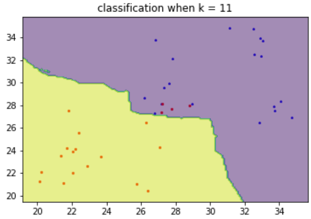
Nearest Neighbours - Introduction

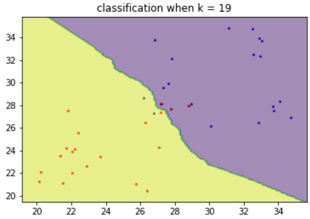
We can get a decision boundary given k:
for example:



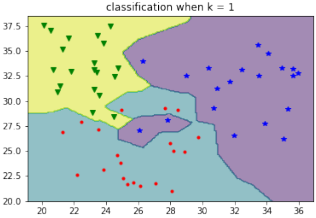
Nearest Neighbours - Introduction

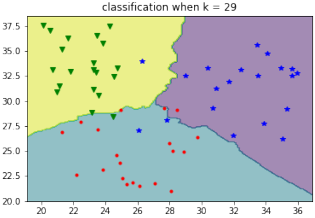
The boundary gets smoother, and generalises better when k is high for example:





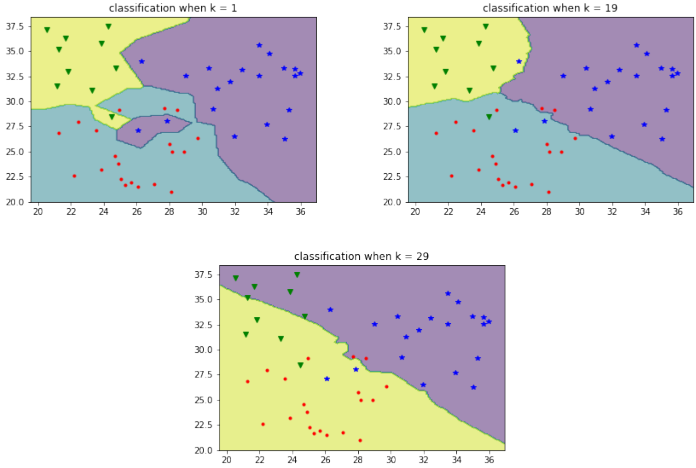
And with multi class classification, equally sized classes:





Nearest Neighbours - Introduction

However, if *k* is too high, where some classes are less common, they can be missed



MNIST ipynb demo

Nearest Neighbours - Introduction

Advantages?

Nearest Neighbours - Introduction

Advantages?

- No assumptions made
- No training phase
- Simple and easy to implement

Problems?

Nearest Neighbours - Introduction

Advantages?

- No assumptions made
- No training phase
- Simple and easy to implement

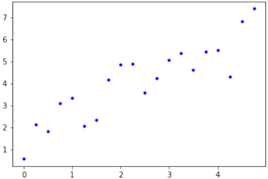
Problems?

- Doesn't scale well with lots of data
- Doesn't scale well with many dimensions

Nearest Neighbours - Regression

KNN can be used to perform regression

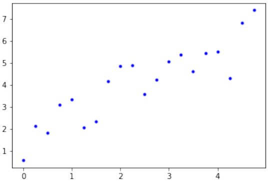
It uses the average value of the k closest data points



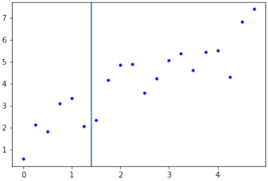
Nearest Neighbours - Regression

KNN can be used to perform regression

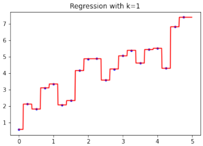
It uses the average value of the k closest data points

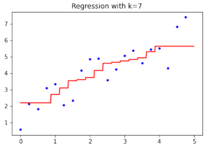


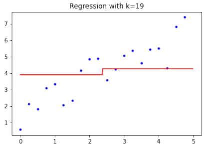
So a point at $x = 1.4$ will have a value ≈ 2 if $k = 1 - 2$



Nearest Neighbours - Regression

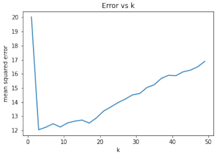






From overfitting to underfitting..

The mean squared errors can be measured for each value of k



Greatest errors at the edges, interpolation easier than extrapolation

Tuning k carefully is important - best done using cross validation

ipynb Height Weight Age regression demo

Nearest Neighbours - Weighted KNN

Up to now, each value in the k nearest neighbours has been treated equally.

Better: if closer neighbours are more important

We can use a range of weighting schemes to do this:

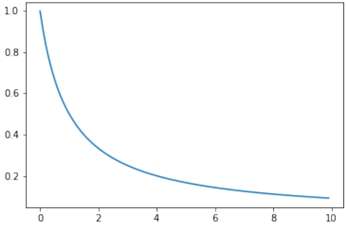
- Inverse Weighting
- Subtraction weighting
- Gaussian Weighting

Nearest Neighbours - Weighted KNN

Inverse Weighting:

$$w = \frac{1}{dist + c}$$

Where c is a constant, avoiding division by zero error if $dist = 0$

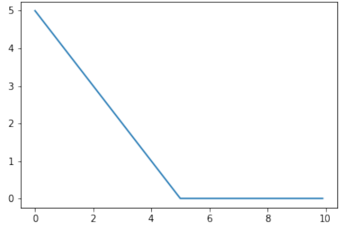


Nearest Neighbours - Weighted KNN

Subtraction Weighting:

$$w = \max(0, c - dist)$$

Where c is a constant

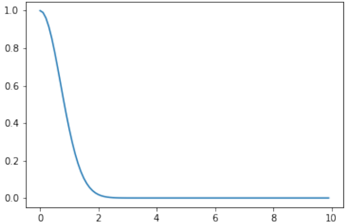


Nearest Neighbours - Weighted KNN

Gaussian Weighting:

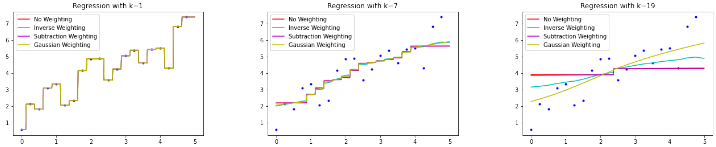
$$w = \exp \frac{-dist^2}{c^2}$$

Where c is a constant

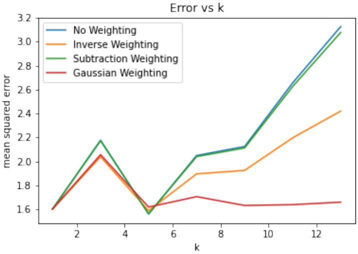


Nearest Neighbours - KNN

Weighted Regression:



Gaussian performs best here, especially with higher values of k



Still has greater errors at the edges of the data, interpolation easier than extrapolation

Nearest Neighbours - KNN

Again, to chose the best weighting scheme, measure performance using cross validation.
Problems?

- Heterogenous Data - features with larger ranges have greater effects
- Outliers affect data a good deal, especially for low k
- For larger k, less common classes can get ignored
- Distance metric determines similarity - usually Euclidean, works badly in high D
- Can use Hamming distance for categorical attributes
- Irrelevant data can force otherwise similar data samples to be far apart
- Computationally expensive if there are lots of data, or highly dimensional data

Nearest Neighbours - KNN

Curse of dimensionality:
For low dimensions, the number of points on the edge is very low
E.g. for a line, the outer 1% of a line is 2% of the line (values at $x > 0.99$, and $x < 0.1$)

Nearest Neighbours - KNN

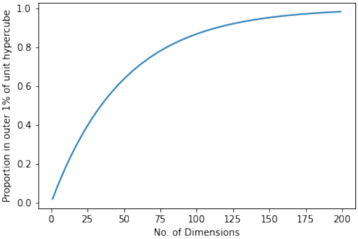
Curse of dimensionality:
For low dimensions, the number of points on the edge is very low
E.g. for a line, the outer 1% of a line is 2% of the line (values at $x > 0.99$, and $x < 0.1$)
For a square, the outer 1% is $1 - 0.98^2 = 0.0396 \approx 4\%$

Nearest Neighbours - KNN

Curse of dimensionality:
For low dimensions, the number of points on the edge is very low
E.g. for a line, the outer 1% of a line is 2% of the line (values at $x > 0.99$, and $x < 0.1$)
For a square, the outer 1% is $1 - 0.98^2 = 0.0396 \approx 4\%$
For a cube, the outer 1% is $1 - 0.98^3 = 0.0588 \approx 6\%$

Nearest Neighbours - KNN

Curse of dimensionality:
For low dimensions, the number of points on the edge is very low
E.g. for a line, the outer 1% of a line is 2% of the line (values at $x > 0.99$, and $x < 0.1$)
For a square, the outer 1% is $1 - 0.98^2 = 0.0396 \approx 4\%$
For a cube, the outer 1% is $1 - 0.98^3 = 0.0588 \approx 6\%$



This means in higher dimensions, data is nearly always extrapolated

Nearest Neighbours - KNN

Curse of dimensionality; For low dimensions, the size of a neighbourhood is small.

e.g. for $k = 10$, number of points $N = 1,000,000$

In a unit line, the average neighbourhood is $\frac{10}{10^6} = 0.00001$ long

Nearest Neighbours - KNN

Curse of dimensionality; For low dimensions, the size of a neighbourhood is small.
e.g. for $k = 10$, number of points $N = 1,000,000$
In a unit line, the average neighbourhood is $\frac{10}{10^6} = 0.00001$ long
In a unit square, the average side length is $\sqrt{\frac{10}{10^6}} = 0.003$ long

Nearest Neighbours - KNN

Curse of dimensionality; For low dimensions, the size of a neighbourhood is small.

e.g. for $k = 10$, number of points $N = 1,000,000$

In a unit line, the average neighbourhood is $\frac{10}{10^6} = 0.00001$ long

In a unit square, the average side length is $\sqrt{\frac{10}{10^6}} = 0.003$ long

In a unit cube, the average side length is $\sqrt[3]{\frac{10}{10^6}} = 0.02$ long

Nearest Neighbours - KNN

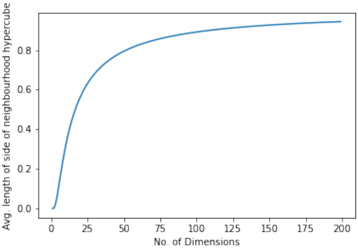
Curse of dimensionality; For low dimensions, the size of a neighbourhood is small.

e.g. for $k = 10$, number of points $N = 1,000,000$

In a unit line, the average neighbourhood is $\frac{10}{10^6} = 0.00001$ long

In a unit square, the average side length is $\sqrt{\frac{10}{10^6}} = 0.003$ long

In a unit cube, the average side length is $\sqrt[3]{\frac{10}{10^6}} = 0.02$ long



This can make it very difficult to work out which are closer, as the distances are nearly all the same

Nearest Neighbours - KNN

Solutions: For heterogenous data?

Nearest Neighbours - KNN

Solutions: For heterogenous data?

- For heterogenous data, can *normalise*

Nearest Neighbours - KNN

Solutions: For heterogenous data?

- For heterogenous data, can *normalise*
- Better to scale factors for each feature to optimise performance
- Could use this to do feature selection
  eg. if works best when scale factor = 0, then feature is useless!

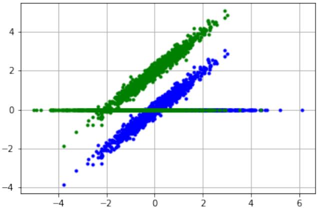
Nearest Neighbours - KNN

Solutions for high D?

Nearest Neighbours - KNN

Solutions for high D?

- Dimensionality reduction.. Care! Some aren't suitable (e.g. MDS, SOM)
- Also.. PCA:



- A random direction could be better!
  Johnson Lindenstrauss lemma:
  if points in a vector space are of high enough dimensionality, they may be projected into a lower dimensional space in a way which approximately preserves the distances between the points, this basis can be generated randomly

Nearest Neighbours - KNN

More solutions for high D?

Nearest Neighbours - KNN

More solutions for high D?

- Use different metric:

Nearest Neighbours - KNN

More solutions for high D?

- Use different metric:
  - Hamming distance for categorical attributes
  - BM25 or TF-IDF for text data
  - Minkowski distance (p-norm) - generalisation of Euclidean distance
  - Kullback - Liebler Divergence for histograms

Nearest Neighbours - KNN

Solutions for lots of data?

- Need to quickly find the nearest neighbour to a particular point in a highly dimensional space

Nearest Neighbours - KNN

Solutions for lots of data?

- Need to quickly find the nearest neighbour to a particular point in a highly dimensional space
  - Could index points in a tree structure?
  - Could hash the points?
  - Could break up the space

Nearest Neighbours - K-D trees

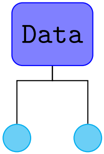
K-D trees are binary tree structures that partition the space along an axis-aligned hyperplane

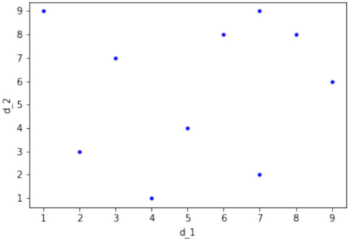
- Chose random dimension
- Divide along median value
- Repeat until depth limit reached or certain number of items in each leaf

Nearest Neighbours - K-D trees

For a simple dataset:

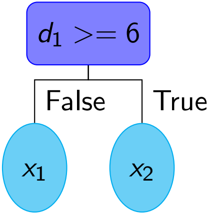
Tree:



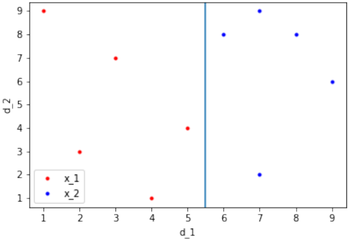


Nearest Neighbours - K-D trees

Tree:

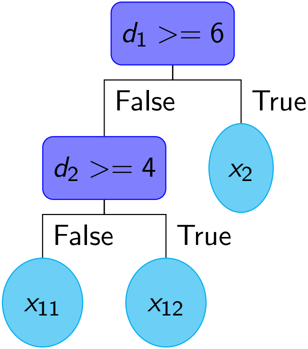


Split:

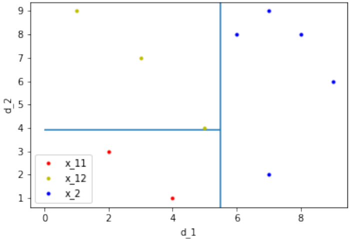


Nearest Neighbours - K-D trees

Tree:

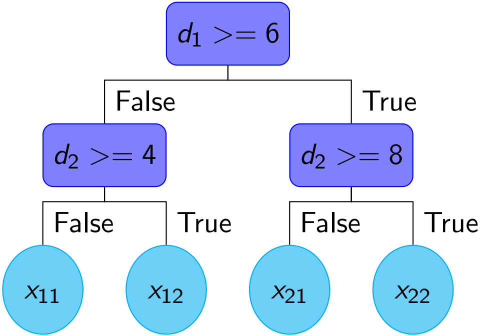


Split:

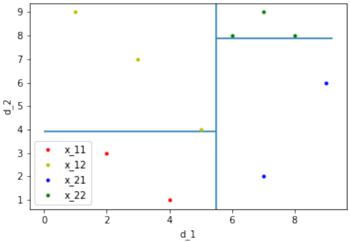


Nearest Neighbours - K-D trees

Tree:



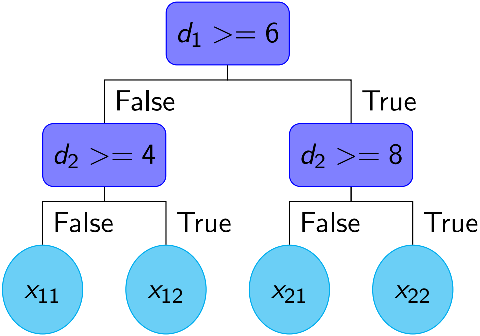
Split:



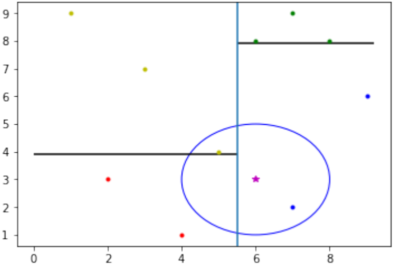
Nearest Neighbours - K-D trees

To Classify: (6, 3)

- Go to correct part of tree and search subspace
- If the border is closer than k-neighbours in the subspace:
  - Go back up tree and search



Split:



Nearest Neighbours - K-D trees

Problems?

- Doesn't scale well to high dimensions
- Often need to search much of the tree
- Need *many* more examples than there are dimensions, at least 2^n
- There are approximate versions, not guaranteed exact answer but do scale
  - Based on ensembles of trees with a randomised split dimension

Nearest Neighbours - LSH

Locality Sensitive Hashing
Makes hash codes that are similar for similar vectors

- Similar items map to the same buckets with high probability
- number of buckets much smaller than number of data samples
- Aims to maximise the probability of a collision for similar items

Nearest Neighbours - LSH

Accomplished by:

- Chose random hyperplanes (h_1, h_2, ..., h_k)
- Each hyperplane with split the space in to 2 regions
- $\therefore$ the space will be sliced in to 2^k regions (buckets)
- Giving a simple code for each of the data points, depending on which side of each hyperplane the data points are
- The same code for each of the data points within the same region
- Compare new point only to training points in the same region
- Repeat with different random hyperplanes (h_1, h_2, ..., h_k)

on board

Gives low complexity , $\approx O(d \log n)$, as compare new data to only $\frac{n}{2^k}$

Nearest Neighbours - Summary

KNN can be used for regression as well as classification

- Using weighting can improve performance
- Poor performance with large data sets
- Can use K-D trees to help overcome these issues
  - Still can have issues with highly dimensional data
  - Often not much improvement in performance
- Curse of dimensionality
  - Affects neighbourhood size
  - Affects amount of extrapolation
- Can use dimensionality reduction to help (but be careful!)
- Fast approximate Nearest Neighbourhood methods - LSH

Also: *Final presentation does not need to be for full coursework, it is to show what you have done so far*