Nearest Neighbour Method

Mingjun Zhong

Department of Computing Science
University of Aberdeen

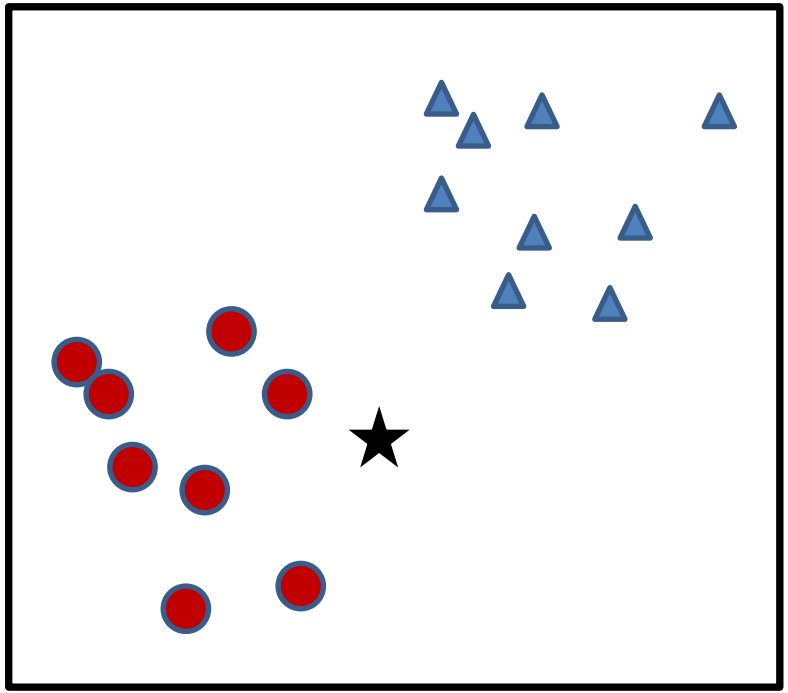
Outline

- Nearest neighbour method
  - Classification and regression
  - Practical issues: distance, k, ties, missing values
  - Optimality and assumptions

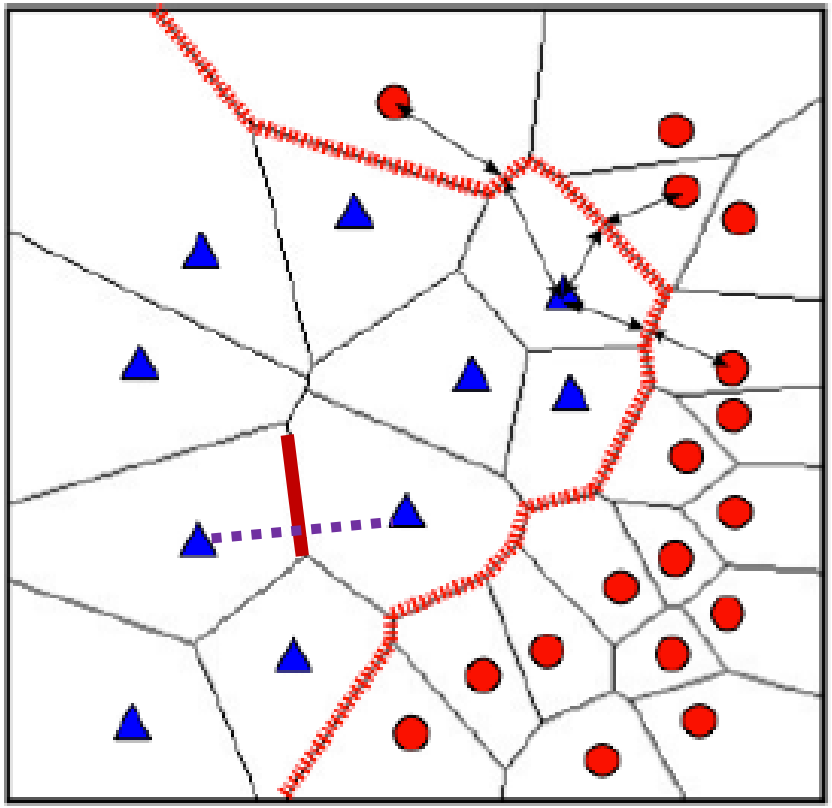
- Why should we normalise or standardise features?

Intuition for kNN

- Set of points (*x, y*) – two classes
- Which class is the star in?
- How would you do it?
- Nearby points are circles – use this as a basis for a learning algorithm



Nearest neighbour classification



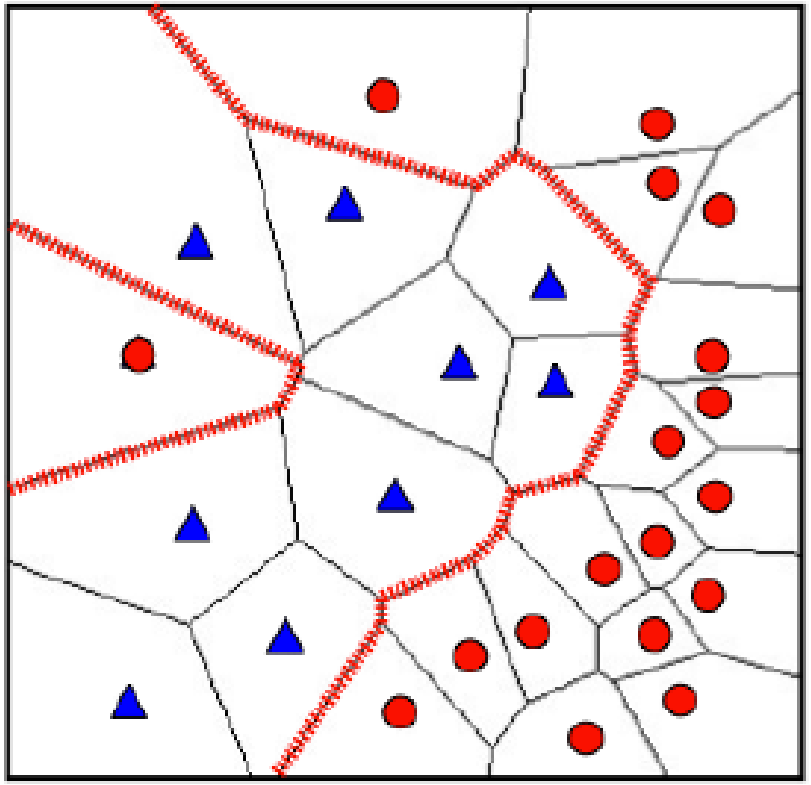
- Use the intuition to classify a new point *x*:
  - Find the most similar training example *x*
  - Predict its class *y*
- *Voronoi* partition
  - Partitions space into regions
  - Boundary: points at same distance from two different training examples
- Classification boundary
  - Non-linear, reflects classes very well
  - Impressive for simple method

Nearest neighbour algorithm

- **Given**:
  - Training examples $\{x_i, y_i\}$: x_i are attribute-value representation of examples; y_i class labels.
  - Testing point x_* that we want to classify
- **Algorithm**:
  - **Step 1**: compute distance from x_* to every training example x_i: $D(x_*, x_i)$.
  - **Step 2**: select the closest instance(x', y')
  - **Step 3**: output the class $y_* = y'$

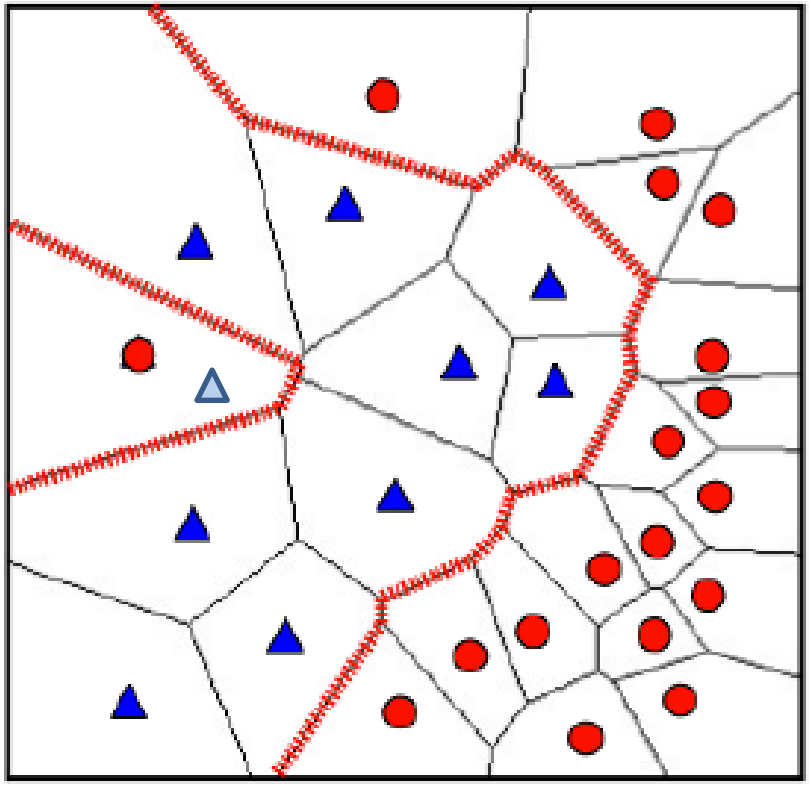
Nearest neighbour: outliers

- Algorithm is sensitive to outliers
  - Single mislabelled example dramatically changes boundary
- Idea:
  - Use more than one nearest neighbour to make decision
  - Count class labels in ***k*** most similar training examples



Nearest neighbour: outliers

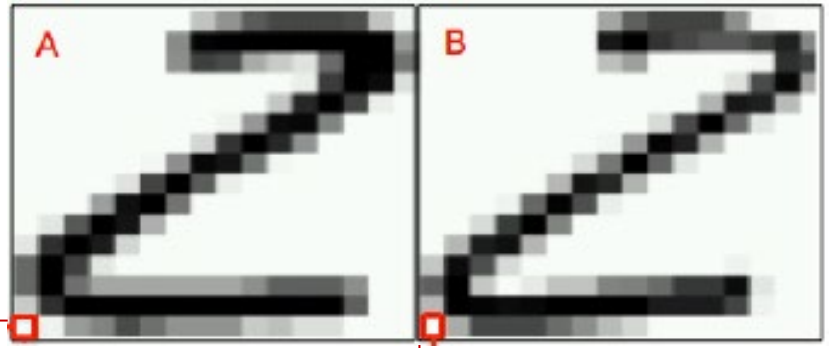
- Algorithm is sensitive to outliers
  - Single mislabelled example dramatically changes boundary
- Idea:
  - Use more than one nearest neighbour to make decision
  - Count class labels in ***k*** most similar training examples



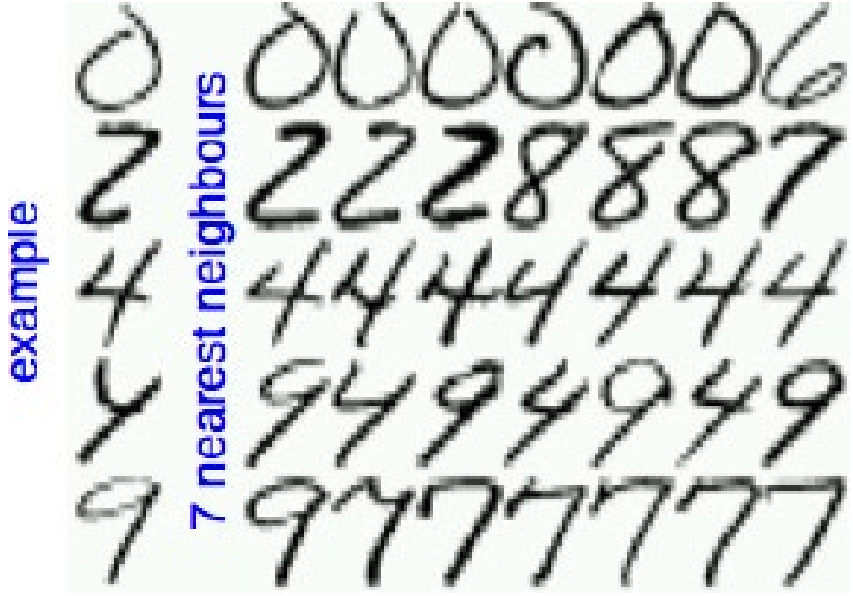
KNN classification algorithm

- **Given**:
  - Training examples $\{x_i, y_i\}$: x_i are attribute-value representation of examples; y_i class labels.
  - Testing point x_* that we want to classify
- **Algorithm**:
  - **Step 1**: compute distance from x_* to every training example x_i: $D(x_*, x_i)$.
  - **Step 2**: select k closest instances $(x_1, y_1), (x_2, y_2), \cdots, (x_k, y_k)$
  - **Step 3**: output the class y_* which is the most frequent class (the majority class) in $(x_1, y_1), (x_2, y_2), \cdots, (x_k, y_k)$

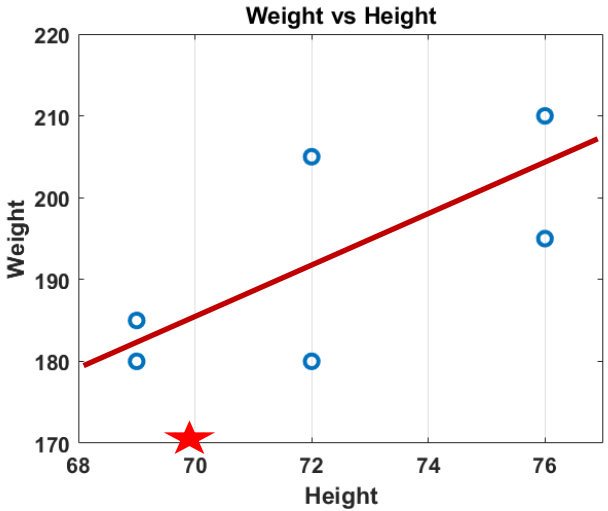
Example: handwritten digits



- 16x16 grayscale bitmaps
- Euclidian distance:
  - On raw pixels
  - $D(A,B) = \sqrt{\sum_r \sum_c \left(A_{r,c} - B_{r,c}\right)^2}$
- Accuracy:
  - K=7: 95.2%
  - Humans: 97.5%



KNN regression algorithm



- Given:
  - Training examples $\{x_i, y_i\}$: x_i are attribute-value representation of examples; y_i real-valued targets.
  - Testing point x_* that we want to predict the target

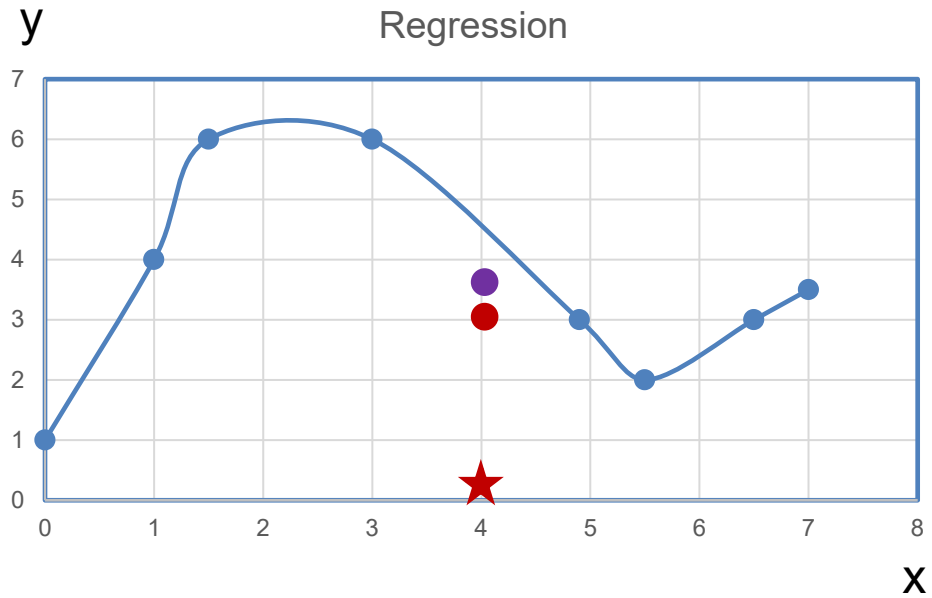
- Algorithm:
  - Step 1: compute distance from x_* to every training example x_i: $D(x, x_i)$.
  - Step 2: select k closest instances $(x_1, y_1), (x_2, y_2), \cdots, (x_k, y_k)$
  - Step 3: output the target $y_* = \frac{1}{k}\sum_{j=1}^{k} y_j$

Example:

- Non-linear regression:
  - $y = f(x)$
- Given:

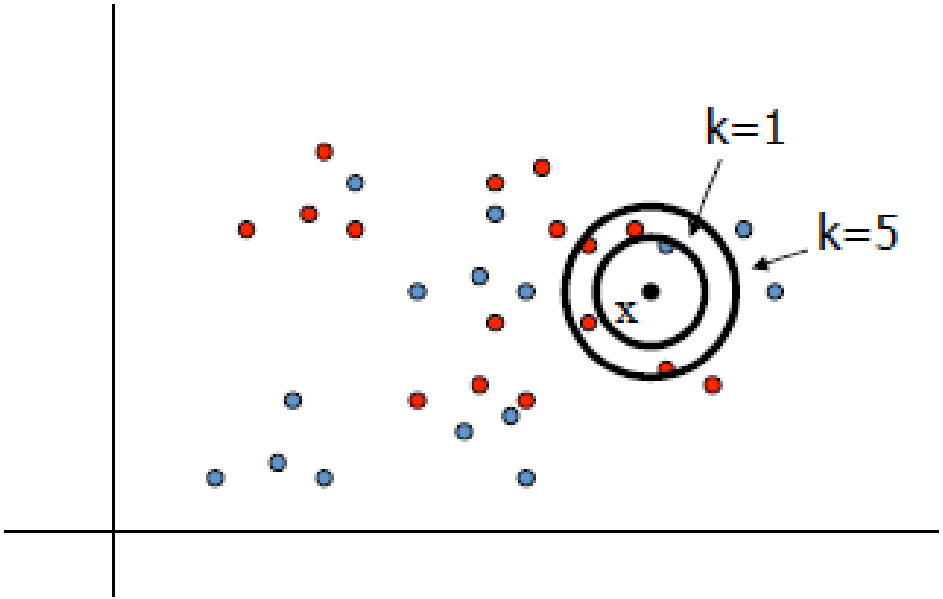
Training Examples								
x	0	1	1.5	3	4.9	5.5	6.5	7
y	1	4	6	6	3	2	3	3.5

- Input $x = 4$, predict target y
  - 1-NN: $y = 3$
  - 3-NN: $y = \frac{11}{3}$



Choosing the value of k

- Value of ***k*** has strong effect on kNN performance
  - Small value: unstable decision boundaries
  - Large value: everything classified to the most probable class
  - Affects smoothness of boundary
- Selecting the value of k (Cross Validation)
  - Set a side a portion of the training data (validation set)
  - Vary ***k***, train model, validation error
  - Pick ***k*** which gives the best performance on validation set



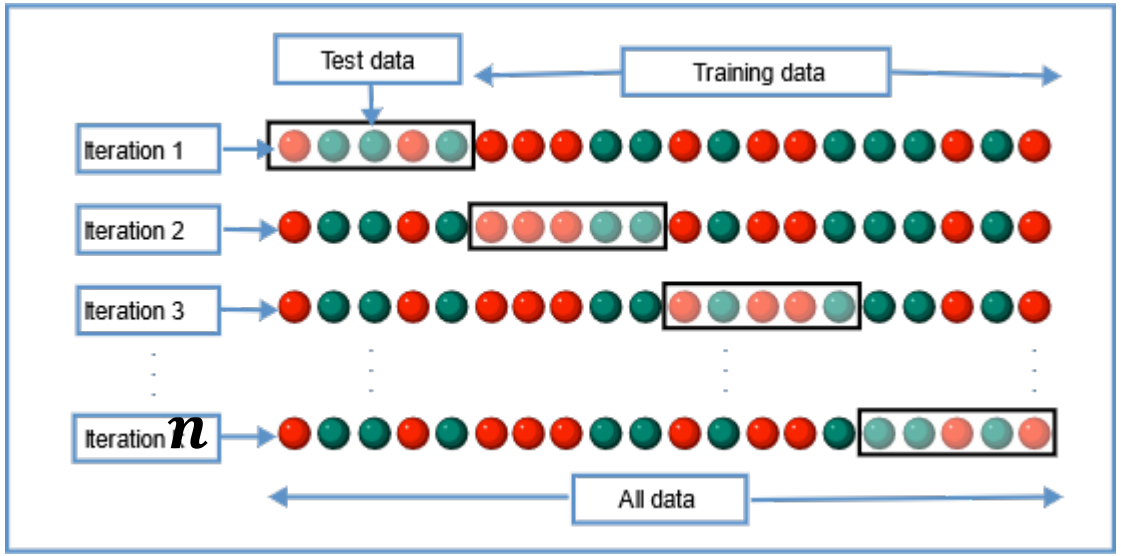
Choosing the value of k

- Using Cross-Validation method:
  - The k in KNN is a hyper-parameter, which influences the performance
  - Each k defines one KNN model
  - Cross-validation (CV) is a model selection method

Choosing the value of k

- What is cross-validation?
- Details to be introduced in later lectures

- n-fold cross-validation:
  - ✓ Shuffle the data samples
  - ✓ Equally split the data samples into n groups
  - ✓ For each k:
    - Iteratively use each group as the test data for validating the model, while using the rest data for training the model
    - In each iteration, compute the accuracy on the test data
    - Average the test accuracy over n iterations
  - ✓ Choose the k with the best average test accuracy.



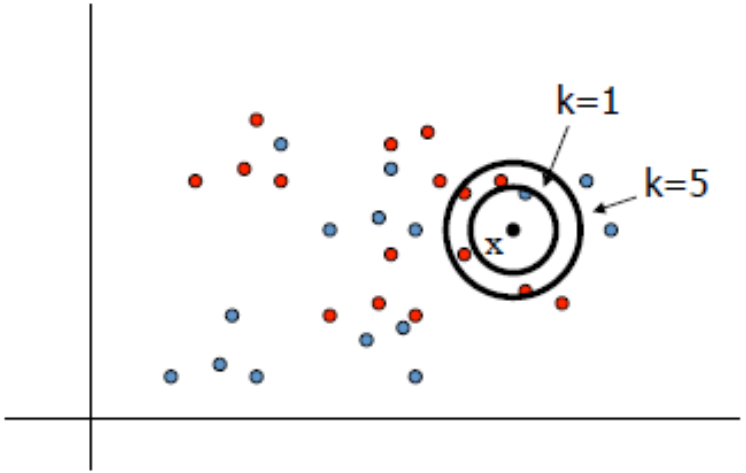
Source: Wikipedia

Distance measures

- Key component of kNN
  - Defines which examples are similar; which are different
  - Has strong effect on performance

- Numeric attributes: Euclidian distance
  - $\boldsymbol{D(x, x') = \sqrt{\sum d(x_d, x'_d)^2}}$
  - Symmetric, spherical, treats all dimensions equally

- Categorical attributes: Hamming distance
  - $\boldsymbol{D(x, x') = \sum_d 1_{x_d \neq x'_d}}$
  - Counts the number of different attributes



- D(0110100, 1010101) = 3
- D("tube", "cube") = 1

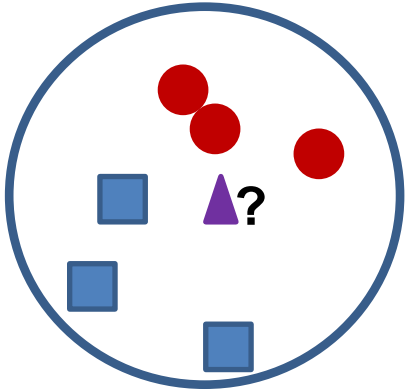
Distance measures

- Minkowski distance:
  - $\boldsymbol{D(x, x') = (\sum d(x_d, x'_d)^p)^{1/p}}$
- Mahalanobis distance:
  - $\boldsymbol{D(x, x') = \sqrt{(x - x')^T S^{-1} (x - x')}}$
  - $\boldsymbol{S}$ is the covariance matrix
- Kullback-Leibler (KL) divergence:
  - $\boldsymbol{x_d > 0, \sum_d x_d = 1}$
  - $\boldsymbol{D(x, x') = -\sum_d x_d log \frac{x_d}{x'_d}}$
  - $\boldsymbol{D(x', x) = -\sum_d x'_d log \frac{x'_d}{x_d}}$
  - asymmetric
- Numeric attributes: Euclidian distance
  - $\boldsymbol{D(x, x') = \sqrt{\sum d(x_d, x'_d)^2}}$
  - Symmetric, spherical, treats all dimensions equally
- Categorical attributes: Hamming distance
  - $\boldsymbol{D(x, x') = \sum_d 1_{x_d \neq x'_d}}$
  - Counts the number of different attributes

Practical issues

- Resolving ties:
  - Equal number of positive and negative neighbours
  - Use odd ***k***: does not solve multi-class
  - Breaking ties:
    - Random: flip a coin
    - Prior: pick the most numerous class
    - Nearest: use *1-nn* to decide

- Missing values
  - Have to ‘fill in’, otherwise cannot compute distance
  - Reasonable choice: average value across entire dataset



Strengths and weaknesses

- **Strengths**
  - Almost no assumptions about the data
  - Smoothness: nearby regions of space assigned to same class
  - Assumptions only made by distance function
  - Non-parametric approach: “let the data speak for itself”
    - Nothing to infer from the data, except ***k*** and distance measure
    - Easy to update in online setting: just add new item to training set
- **Weaknesses**
  - Need to handle missing values: fill-in or create a special distance
  - Sensitive to class-outliers: mislabelled training instances
  - Sensitive to lots of irrelevant attributes
  - Computational expensive:
    - N: number of training examples; d: dimension of attributes
    - N grows: system will become slower and slower
    - Expensive in testing, not training time (too bad)

Summary

- Key idea: nearby points assigned to same class
- Can be used for classification and regression
- Simple, non-linear, non-parametric, asymptotically optimal
- Select k by cross validation
- Naïve implementations slow for big data

Normalising or Standardising Data Features

Normalisation or Standardisation

- Normalisation: scaling features to a range [a, b], e.g., [0, 1]
  - Divide each feature value by the feature's norm value.

- Standardisation: make each feature have mean value = 0 and standard deviation = 1.
  - $x = (x_1, x_2, \cdots, x_N)$: the data sample
  - $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$: the mean value of x
  - $s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$: the standard deviation of x
  - These all can be calculated using Python

Normalisation or Standardisation

- Why is normalisation or standardisation important?
- Most of the models need to do such preprocessing before training:
  - Linear regression, logistic regression, k-nearest neighbour
- Why? :
  - Interpretation: making the feature units normalised
    - ✓ House sizes: **1300 and 1000;** numbers of bedrooms: **4 and 2**
    - ✓ Normalised or standardised features can interpret models
  - Make the learning process converge faster
    - ✓ Feature values which are not normalised make the Gradient Descent algorithm hard to converge (*have you tried last week's practical?*)
    - ✓ KNN algorithm: distance-based, if not normalised, distances are biased to those with large scales
      - ❖ $\boldsymbol{D(x, x') = \sqrt{\sum d\left(x_d, x'_d\right)^2} = \sqrt{(1300 - 1000)^2 + (4 - 2)^2}}$
      - ❖ The effect of number of bedrooms is minor.