THE UNIVERSITY OF MELBOURNE

Instance-based Learning

Semester 1, 2021

Ling Luo

Outline

- Instance-based Learning
- Comparing Instances
  - Similarity metrics for feature vectors
- Nearest Neighbour Classification

Instances

The input to a machine learning system consists of:

- **Instances**: the individual, independent examples of a concept
- **Attributes**: measuring aspects of an instance
- **Labels/Classes**: things that we aim to learn

- Each instance is described by n attribute-value pairs.
- Each instance also has a class label.

Instance-based learning (1)

- Supervised learning algorithms: learn from labelled examples
  - Input: instances
  - Model: Some kind of function that maps instances to class labels
- Instance-based learning: belongs to supervised learning algorithms
  - Requires labelled examples stored in memory
  - Learns directly by example

memory-based learning

Instance-based learning (2)

- Example: Cool/Cute Classifier

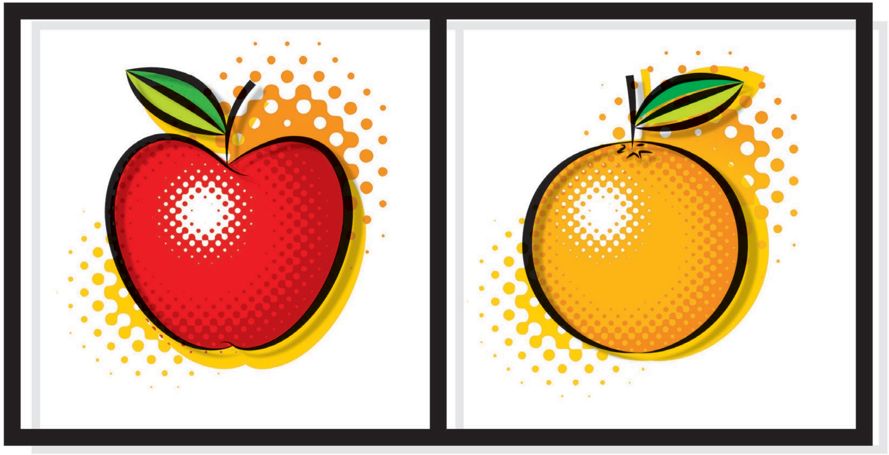
Entity	Class
baby	Cute
sports car	Cool
tiger	Cool
Hello Kitty	Cute
water	???

How would we predict the class for the following instances?

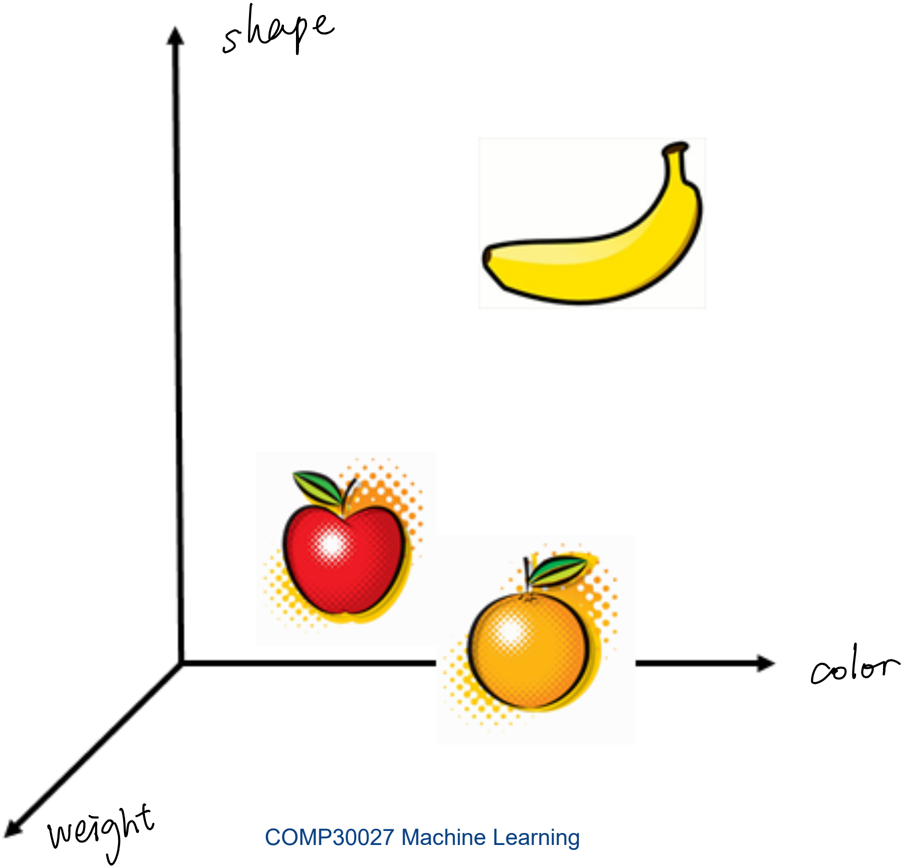
train, koala, book

Comparing Instances

Compare and Contrast (1)



Compare and Contrast (2)



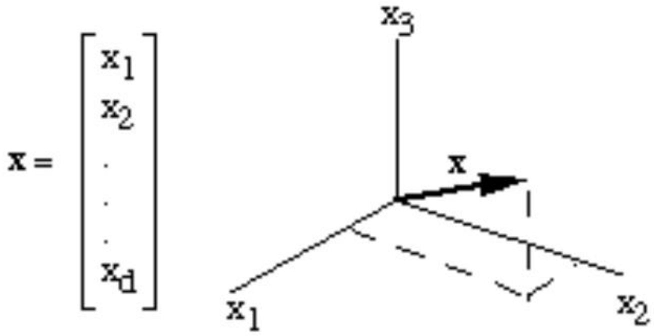
Feature Vectors (1)

- A feature or attribute is any distinct aspect, quality, or characteristic of an instance.
- A feature vector is an *n*-dimensional vector of features that represent that instance.

- Features may be
  - nominal/categorical/discrete (e.g. colour, gender)
  - ordinal (e.g. temperature: cool < mild < hot)
  - numeric/continuous (e.g. height, age)

Feature Vectors (2)

- A vector locates an instance (object, document, person, . . .) as a point in an *n*-dimensional space.
- The angle of the vector in that space is determined by the relative weight of each term.



Feature Vector Vector Space

Similarity and Dissimilarity (1)

- Similarity
  - Numerical measure of how alike two data objects are
  - Higher when objects are more alike
  - Often falls in the range [0,1]
- Dissimilarity
  - Numerical measure of how different are two data objects
  - Lower when objects are more alike
  - Minimum dissimilarity is often 0
  - Upper limit varies

Similarity and Dissimilarity (2)

- Comparing two data objects with attribute values p and q

Attribute Type	Dissimilarity (all 3 are distance measure)	Similarity
Nominal	$d = \begin{cases} 0 & \text{if } p = q \\ 1 & \text{if } p \neq q \end{cases}$	$s = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases}$
Ordinal (limitation assume evenly separated eg temp low medium high very high might not be true)	$d = \frac{\lvert p-q \rvert}{n-1}$ range $\in [0,1]$ measured by interval length (values mapped to integers 0 to $n-1$, where n is the number of values)	$s = 1 - \frac{\lvert p-q \rvert}{n-1}$
Interval or Ratio	$d = \lvert p - q \rvert$ $[0, +\infty)$	$[0,1]$ $s = -d$, $s = \frac{1}{1+d}$ or $s = 1 - \frac{d - min\_d}{max\_d - min\_d}$ normalise to make $\in [0,1]$

Distance Measures

one type of dissimilarity

- A distance measure is a function that takes two points in a space as arguments.
  - No negative distances
$$d(x, y) \geq 0$$
  - Distance is zero from a point to itself
$$d(x, y) = 0 \text{ if and only if } x = y$$
  - Distance is symmetric
$$d(x, y) = d(y, x)$$
  - Triangle inequality
$$d(x, y) \leq d(x, z) + d(z, y)$$

Distance Measures

Example: dissimilarity measure that is not a distance metric

- Given two sets A and B, $A - B$ is defined as the set of elements of A that are not in B
- Define $d(A, B) = size(A - B)$

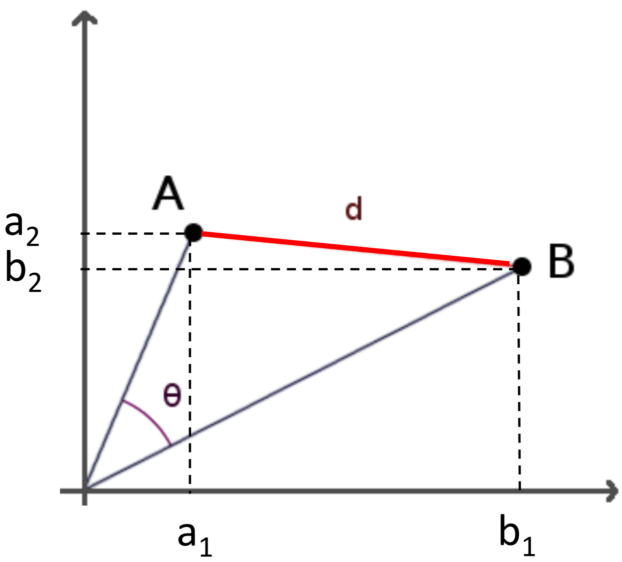
- If $A = [1,2,3,4]$, $B = [2,3,4]$
- $d(A, B) = ?$ $d(B, A) = ?$

1

ϕ.

Euclidean Distance (L2 distance)

- Given two items *A* and *B*, and their feature vectors ***a*** and ***b***, their distance ***d*** in Euclidean space is:

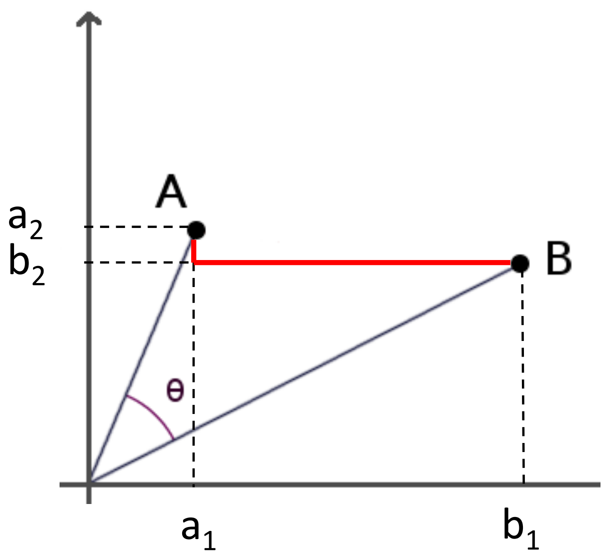


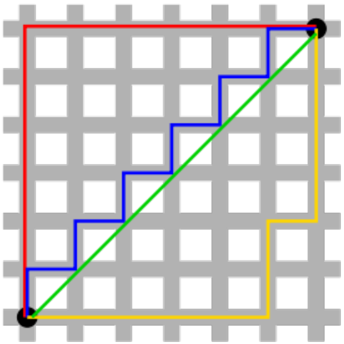
In *n*-dimensional space:

$$d(A,B) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

Manhattan Distance (L1 distance)

- Given two items *A* and *B*, and their feature vectors ***a*** and ***b***, their Manhattan distance ***d*** is:





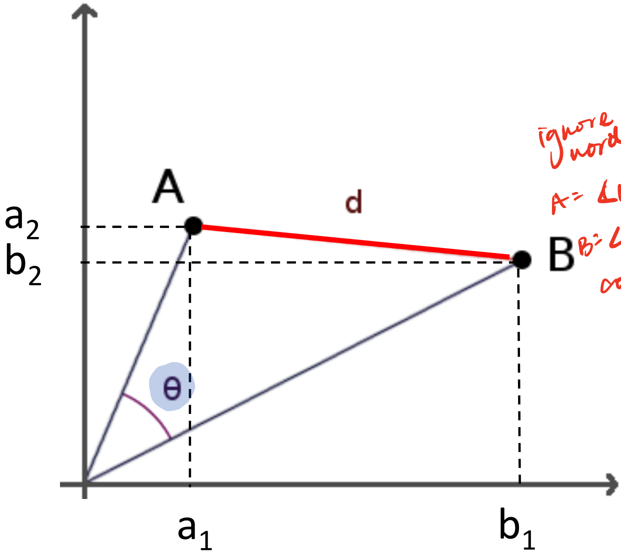
In *n*-dimensional space:

$$d(A,B) = \sum_{i=1}^{n} |a_i - b_i|$$

Cosine Similarity

- Given two items *A* and *B*, and their feature vectors ***a*** and ***b***, we can calculate their similarity via the cosine of the angle θ between vectors ***a*** and ***b***

similarity between neighbour vector of word.



ignore word length

A= <Hi>

B= <Hi, Hi, Hi>

cos(A,B) = 1

In *n*-dimensional space:

ignore the length

$$\cos(A,B) = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}}$$

normalise by length

eg

a = (1,1,1)

b = (2,2,2)

cos (A,B) = 1

Nearest Neighbour Classification

Nearest Neighbour Classification

- What is a Nearest Neighbour?

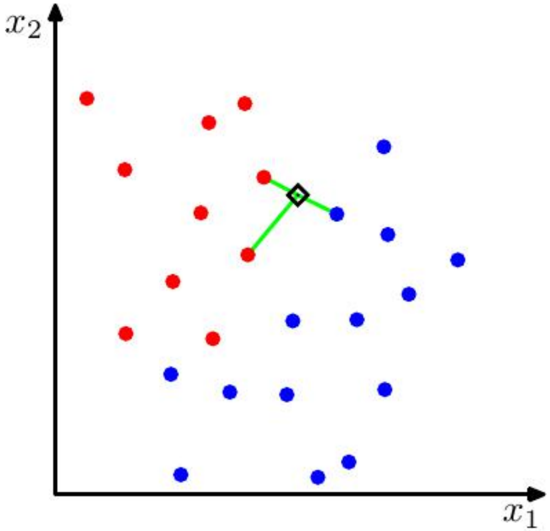
- The closest point: max similarity or min distance

$$d(x, y) = \min(d(x, z) | z \in Y)$$

where Y contains all the neighbours of x.

Nearest Neighbour Classification

- Given class assignments of existing data points, classify a new point (black)

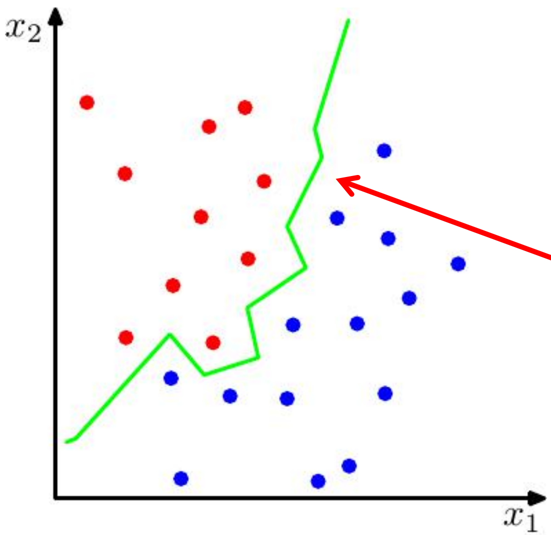


(a)

Consider the class membership of the closest data point

Nearest Neighbour Classification

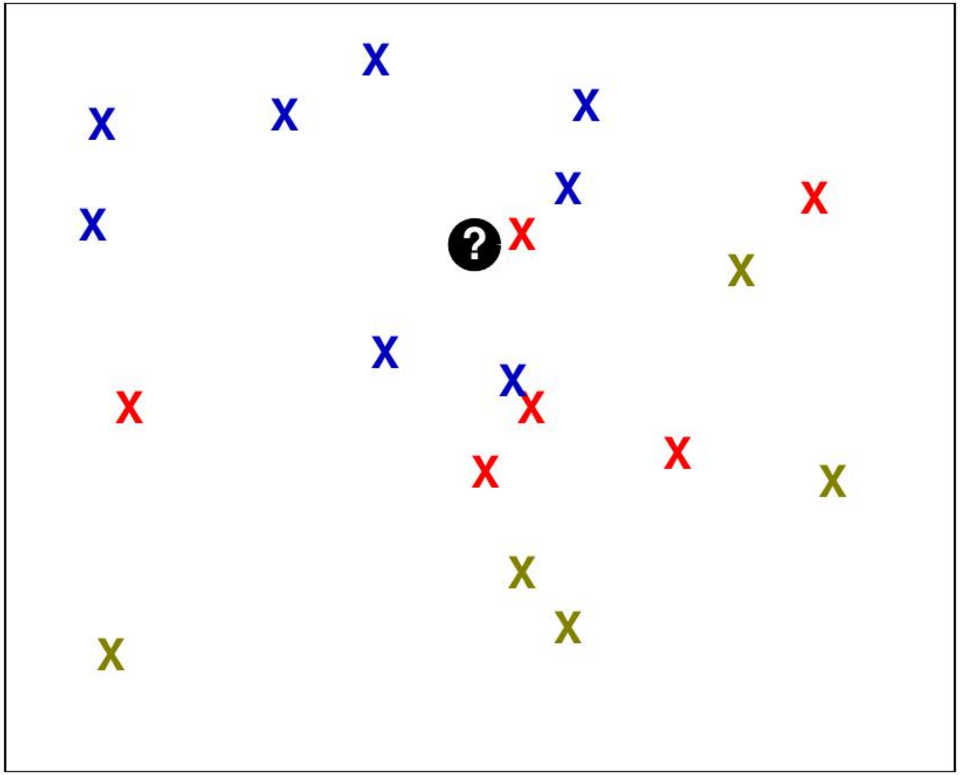
- Given class assignments of existing data points, classify a new point (black)



(b)

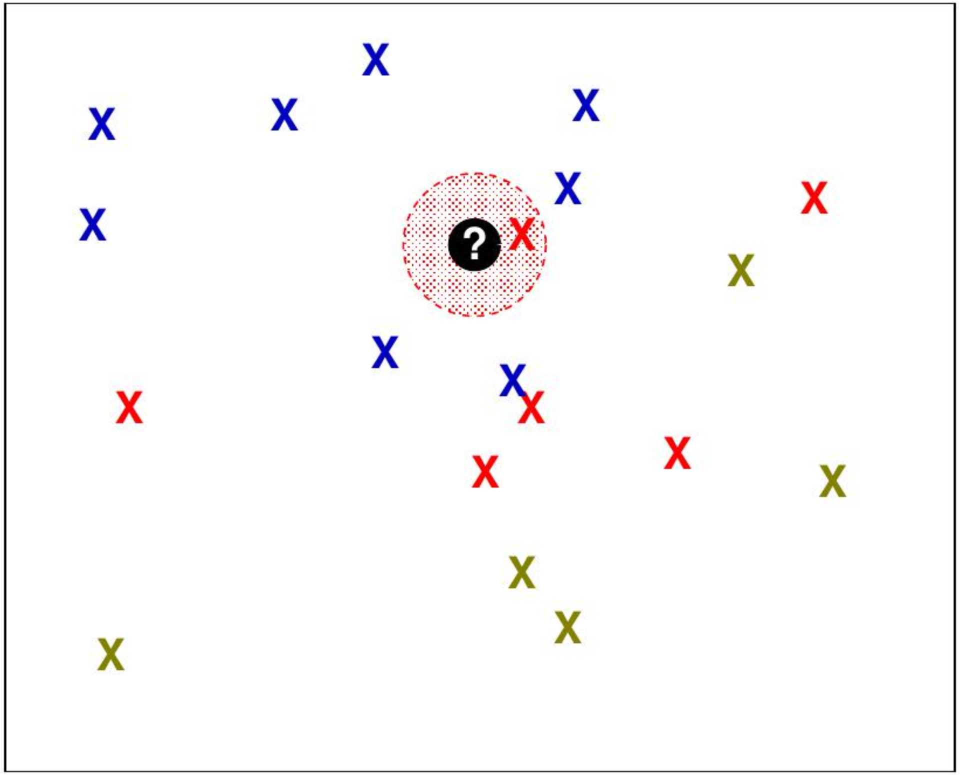
the induced **decision boundary** using nearest neighbour classification

K Nearest Neighbours



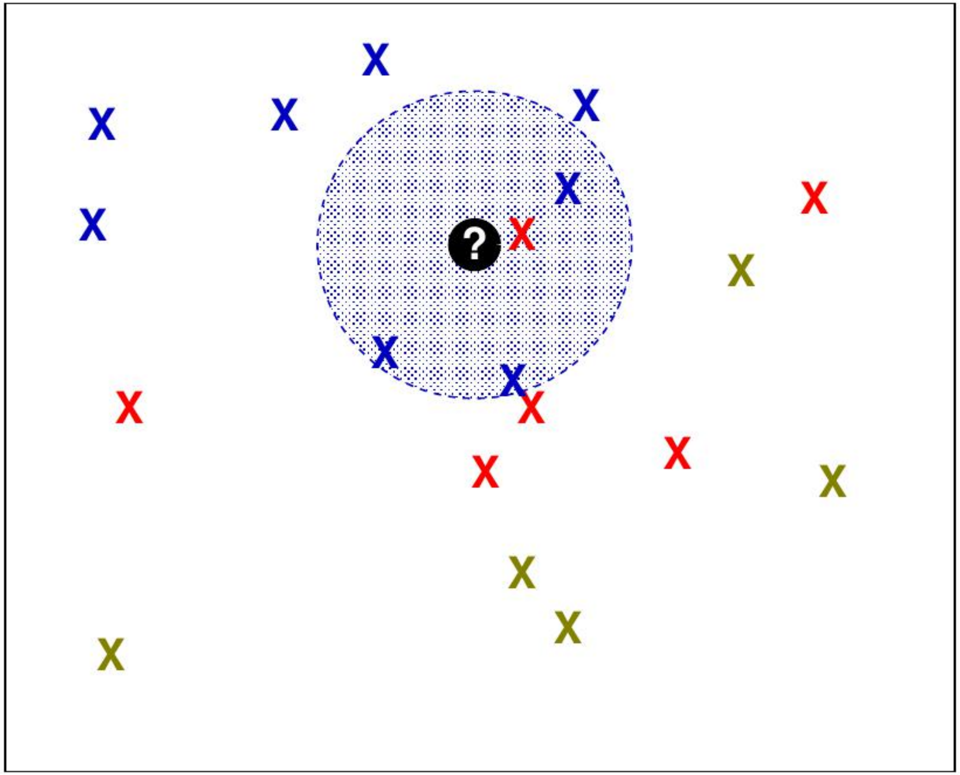
K Nearest Neighbours

K = 1



K Nearest Neighbours

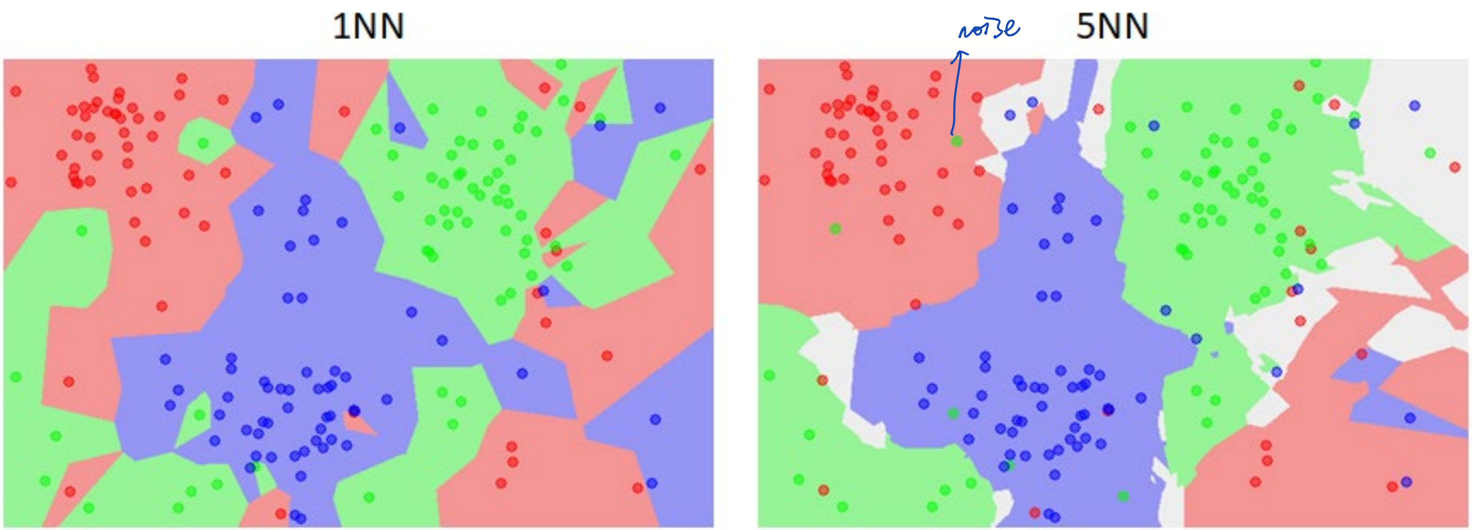
K = 4



Choosing K

- Smaller values of *K* tend to lead to lower performance due to noise (overfitting) sensitive to noise
- Larger values of *K* tend to drive the classifier performance toward Zero-R performance
- Density of the data points tend to be sparse in high dimension curse of dimensionality
  neighbour is far away
- Generally trial and error over the training data is the only way of getting suitable *K*

Choosing K



Source: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Choosing K

- *K* is generally set to an odd value ... why?
- Breaking Ties: If there are equal numbers of votes for multiple classes, we need some tie breaking mechanism
  - random tie breaking
  - take the class with the highest prior probability

    based on distribution of class
  - see if the addition of the (k+1)-th instance(s) breaks the tie

NN Classification Variants

- **K-NN**: Classify the test input according to the majority class of the *K* nearest training instances.

- **Weighted K-NN**: Classify the test input according to the weighted accumulative class of the *K* nearest training instances, where weights are based on similarity of the test input to each of the *K* neighbours.

closer → more weight

Weighting Strategies

Weight the vote of each instance:

- **Equal** weight: classify according to the **majority class** of K neighbours

$\frac{1}{k} \sum_{n \in N_k(x')} y_n$

- By the **inverse linear distance** from the test instance to instance j

$d_j \uparrow$ $w_j \downarrow$. normalise to $[0,1]$

$$w_j = \frac{d_{max} - d_j}{d_{max} - d_{min}}$$

$\frac{1}{k} \sum_{n \in N_k(x')} w_n y_n.$ KNN set to x'

where d_{min} is for the nearest neighbour of the test instance, and d_{max} is for the furthest neighbour of the test instance

- By the **inverse distance** from the test instance to instance j

$$w_j = \frac{1}{d_j + \epsilon}$$

$\to$ adjust to avoid $\frac{1}{0}$

Weighting Strategies: Example

- What is the class label using different weighting strategies?

Instance	Class	Distance
d_1	no	0
d_2	yes	1
d_3	yes	1.5
d_4	yes	2

- **Equal weight** (majority voting):
  yes = 3
  no = 1
- **Inverse linear distance** voting:
  $d_{min} = 0$, $d_{max} = 2$,
  yes = $(\frac{1}{2} + \frac{0.5}{2} + 0) = \frac{3}{4}$
  no = 1
- **Inverse distance** voting ($\epsilon = 0.5$)
  yes = $(\frac{1}{1.5} + \frac{1}{2} + \frac{1}{2.5})$ =1.57
  no = $\frac{1}{0.5} = 2$

Implementation

- A typical implementation involves the *brute-force* computation of distances between a test instance and every training instance.
  - For *N* training instances in *D* dimensions, this approach costs $\mathcal{O}(DN)$
  - Efficient brute-force searches can be very competitive for small datasets
  - However, as the number of samples *N* grows, the brute-force approach quickly becomes infeasible

time consuming

Comparison

KNN. NB. DT all inductive learning
but KNN is instance-based learning

- The model built by K-NN is the dataset itself:
  - K-NN is lazy
  - The time we save in training is lost if we have to make many predictions

time consumed in different phases

- The model built by Naive Bayes/Decision Trees is generally much smaller than the dataset:
  - Given *C* classes and *D* attributes, predicting the class of a test instance requires approximately $\mathcal{O}(CD)$ calculations for Naive Bayes, and $\mathcal{O}(D)$ node traversals for a Decision Tree

not instance learning

need more time to build model
but once the model is built
easy to classify

Strengths and Weaknesses

- Strengths

  kmeans and kNN ↑ supervised
  ↑ unsupervised
  classify into k groups

  - Simple, instance-based, model free
  - Can produce flexible decision boundaries
  - Incremental (can add extra data on the fly)
- Weaknesses
  - Requires a useful distance function
  - Arbitrary *K* value
  - Lazy learner: everything is done at run time
  - Prone to noise and the curse of high dimensionality

Summary

- Representing instances as feature vectors
- Measuring distance and similarity
- What is K-NN, and why do we call it an instance-based learning method?
- What parameters do we have to choose for K-NN?

K-means
unsupervised method

KNN
supervised method.

both rely on distances.

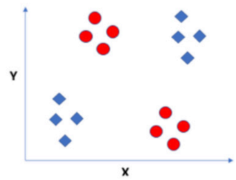
both need to set K

- Readings: Tan et al. (2018, 2nd Edition)
  - Similarities: Section 2.4
  - NN classifier: Section 6.3

Question 4

0.33 / 1 pts

Consider the dataset shown below with real-valued features , we would like to train K-NN (K = 1) and Naïve Bayes on this dataset, with 70% data for training and 30% for testing. Which algorithm would likely to have higher accuracy and why? Check all that applies.



Correct Answer

- [ ] Naïve Bayes would likely to underperform as features are not independent

- [ ] Naïve Bayes would perform well compared with KNN due to its strong assumption of condition independence

Correct!

- [x] KNN would perform better than Naïve Bayes as data is nicely clustered

Correct Answer

- [ ] In a worst-case scenario, KNN's performance could suffer for K = 1 when the test set includes all the instances of a cluster.

Naïve Bayes is a linear classifier. There is not a line that we could draw to separate positive instances from negative instances for this dataset. If we split the data in test and training sets, Naïve Bayes would likely to get some test instances wrong, while KNN would perform better as the data is clustered. With K = 1 in KNN, if the test set includes ALL the instances in one of these clusters, the entire cluster would be misclassified.