Eager Learning

- ML algorithms like ID3, C4.5 or Neural Networks are *eager learners*
  - Use a training data set to
  - Generalize rules, induce a tree or a function that can be applied to categorize future inputs
  - Processing time is done up-front before query time
  - After querying they discard any inputs
Eager learning methodology

- Obtain data set
- Identify target (output) attribute (this is what we want to predict)
- Analyse input features
  - Estimate which are predictive of target
  - Are combinations of input features required (e.g., a simple ratio of two inputs)
- Analyse data set and remove noisy items
- Divide data set in training and test sets

Eager learning methodology

- Identify possible ML algorithms based on:
  - Data types (discrete, continuous)
  - Classification or regression task
  - Type of output required
    - Function
    - Decision tree
    - Neural network

Eager learning methodology

- Run algorithm(s) on training data
- Validate on test data
- Better still do 10 fold cross validation
- Tweak parameters of algorithm
- Repeat validation
- Consider using an ensemble of algorithms
Lazy learners

- Lazy learners have three characteristics:
  - They defer all (most) processing until query/run-time
  - They discard any generated functions/answers
  - They retain the query with the stored data

Lazy Learning

![Diagram](N instances) → ML algorithm → classification → (N + 1) instances

Lazy vs. Eager

- Lazy learners have low computational costs at training (~0)
- But may have high storage costs
- High computational costs at query
- Lazy learners can respond well to dynamic data where it would be necessary to constantly re-train an eager learner
Instance-based learners

- store all the training data
- when a new query instance is encountered, a set of related instances are retrieved from memory and used to classify the instance
- can construct a different approximation function of the target function for each distinct query instance

Eager learners

A complex hypothesis
Eager learners
A simple hypothesis

Lazy learners
k-nearest neighbour

Lazy learners
k-nearest neighbour
Lazy learners

k-nearest neighbour

Instance-based learners

- significant advantage
- when the target function is potentially very complex
- but can be described by a collection of simple local approximations
Instance-based learners

- Disadvantages
  - Cost of classifying new instances can be high, so efficiently indexing training instances very important
  - Similarity has to be determined for each attribute or feature

Instance-based learners

- Nearest neighbour (k-NN)
  - Most basic method - all instances are points in an \( n \)-dimensional space
  - Distance is defined as standard Euclidean distance
  - K-NN finds the nearest neighbours to a query in the \( n \)-dimensional space
  - Values may be discrete or real

Nearest Neighbour

\[
\text{Similarity}(T, S) = \sum_{i=1}^{n} f(T_i, S_i) \times w_i
\]

where:
- \( T \) is the target case
- \( S \) is the source case
- \( n \) is the number of attributes in each case
- \( i \) is an individual attribute from 1 to \( n \)
- \( f \) is a similarity function for attribute \( i \) in cases \( T \) and \( S \)
- \( w \) is the importance weighting of attribute \( i \)
Nearest Neighbour

- imagine a decision with two factors that influence it
- should you grant a person a loan?
  - net monthly income
  - monthly loan repayment

these factors can be used as axes for a graph

a previous loan can be plotted against these axes
Nearest Neighbour

- and a second loan

- and more loans

- and even more loans
Nearest Neighbour

- past cases (loans) may form clusters

![Graph showing clusters of loans based on net monthly income and monthly loan repayment.]

Nearest Neighbour

- past cases (loans) may tend to form clusters

![Graph showing clusters of loans with good loans highlighted in green and bad loans highlighted in red.]

Nearest Neighbour

- past cases (loans) may tend to form clusters

![Graph showing clusters of loans with good loans highlighted in green and bad loans highlighted in red.]
Nearest Neighbour

- a new loan prospect can be plotted on the graph

new case

and the distance to its nearest neighbours calculated
Nearest Neighbour

- and the distance to its nearest neighbours calculated

The best matching past case is the closest
Nearest Neighbour

- the best matching past case is the closest

This suggests a precedent

- the loan will be successful
Nearest Neighbour

- over time the prediction can be validated

![Graph showing monthly loan repayment vs net monthly income with a dot labeled as it was a good loan.]

Nearest Neighbour

- the system is learning to differentiate good and bad loans better

![Graph showing monthly loan repayment vs net monthly income with a yellow arrow.]
Nearest Neighbour

- as more cases are acquired its performance improves

\[ \text{Euclidean Distance} = \sqrt{(X_B - X_A)^2 + (Y_B - Y_A)^2} \]

The weight of the X axis (income) is increased.
Nearest Neighbour

- Require a unique similarity function for each attribute or feature (not always a trivial problem) – *local similarity* \( f(T_i, S_i) \)
- Local similarities are combined to give a *global similarity* – \( \text{sim}(T, S) \)
- k-NN Requires every feature of the query to be compared to every feature of every instance/case at run-time
- Not very efficient 😔

Nearest Neighbour

distance weighted k-Nearest neighbour is a highly effective algorithm for many practical problems robust to noisy data if the training set is large enough
- bias is that the classification of an instance is most similar to other instances that are nearby in Euclidean distance
- But then again that's the point

Nearest Neighbour

- because distance is calculated on all attributes - irrelevant attributes are a problem - curse of dimensionality
- some approaches weight attributes to overcome this - stretching the Euclidean space – determined automatically using cross-validation
- alternatively eliminate the least relevant attributes - they used leave-one out cross-validation – ideal for IBL
Nearest Neighbour

- could locally stretch an axis...but more degrees of freedom...so more chance of overfitting...useful if problem space is not uniform...problem of over fitting
- much less common, but it is used in CBR
- efficient indexing of instances can be done with kd-trees (we’ll discuss later)
- possible to pre-compute a position of each instance in the Euclidean space then simply position query in the space

Summary

- IBLs (k-NN is an IBL) delay processing until prediction time they form a different local approximation for each query instance
- can model complex functions by a combination of less-complex local approximations
- information present in the training data is never lost
- can be computationally expense to label new instances
- finding appropriate distance metric can be difficult
- irrelevant attributes can have a negative impact