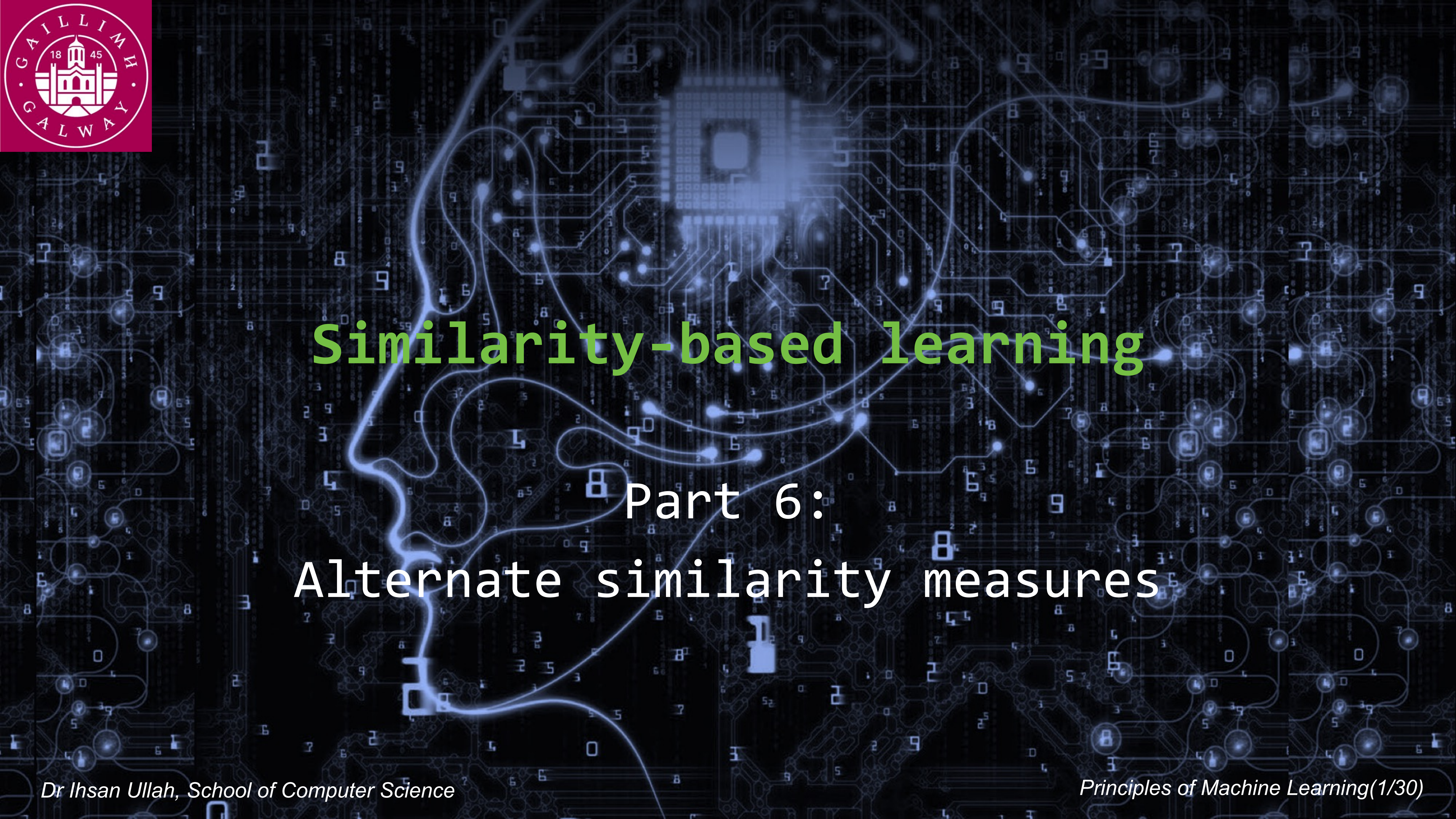


Similarity-based learning

Part 6: Alternate similarity measures

Recap: measuring similarity

- So far, we have measured similarity using distance metrics only
- Each independent variable/attribute is treated as a dimension in hyperspace
- We discussed the well-known Euclidean and Manhattan distance metrics
- We also considered the Minkowski distance, which generalises both Euclidean and Manhattan distances
- In this section we will discuss some issues which can be encountered when measuring similarity and introduce some alternate ways to measure similarity (e.g. similarity indices)
- Note: difference between a similarity index and a distance metric

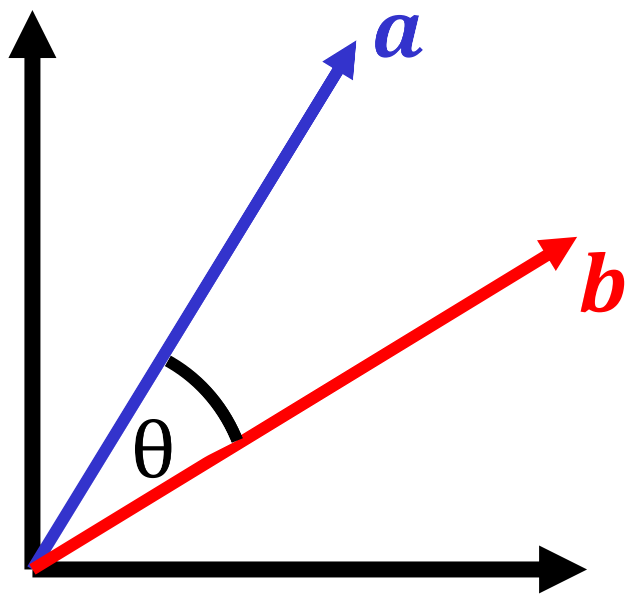
Data normalisation

- Problem — Scaling:
  - Attribute 1 has range 0-10,
    Attribute 2 has range 0-1000
  - Attribute 2 will dominate calculations
- Solution:
  - Rescale all dimensions independently
    - Mean=0, Std deviation=1 [Z-Normalisation] (HousePrices-1NN.xlsx has a worked example)
      D ← (D - Mean) / StDev
    - Min=0, Max=1 [0-1 Normalisation] (also referred to as "range normalisation")
      D ← (D – Min) / (Max – Min) (also utopia = max, nadir=min)

Normalisation is important in many other areas of machine learning and optimisation

Cosine similarity

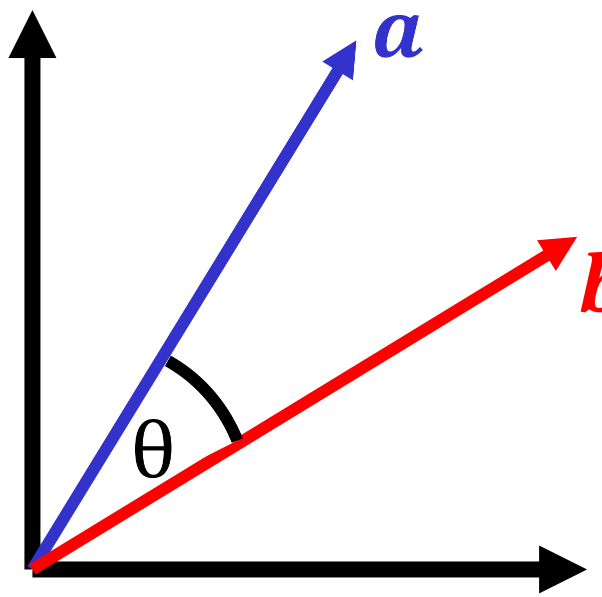
- Cosine similarity is an **index** that may be used to measure the similarity of two instances with continuous attributes. It is the **cosine** of the inner angle between the two vectors that extend from the origin to the points of interest in the feature space
- As before m is the number of features/attributes (i.e. the dimension of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$)
- $\boldsymbol{a} \cdot \boldsymbol{b}$ is the vector dot product, and $|\boldsymbol{a}|$ is the magnitude of the vector $\boldsymbol{a}$



$$Cosine(\boldsymbol{a}, \boldsymbol{b}) = cos(\theta) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}|} = \frac{\sum_{i=1}^{m}(\boldsymbol{a}[i] \times \boldsymbol{b}[i])}{\sqrt{\sum_{i=1}^{m}(\boldsymbol{a}[i])^2} \times \sqrt{\sum_{i=1}^{m}(\boldsymbol{b}[i])^2}}$$

Cosine similarity

- The values of $Cosine(\boldsymbol{a},\boldsymbol{b})$ will be in the range 0 to 1 if all attribute values are non-negative, or -1 to +1 if negative values are present. Can use 0-1 range normalisation to ensure $Cosine(\boldsymbol{a},\boldsymbol{b})$ in 0-1 range.
- $Cosine(\boldsymbol{a},\boldsymbol{b})$ = 1 -> most similar (vectors are parallel)
- $Cosine(\boldsymbol{a},\boldsymbol{b})$ = 0 -> least similar (vectors at 90 deg)
- **Note:** for this index higher values indicate greater similarity (for distance metrics the opposite is true)
- $Cosine(\boldsymbol{a},\boldsymbol{b})$ allows **scale-invariant** comparisons between instances



cos(0) = 1.0

cos(90) = 0.0

Similarity for discrete attributes

- So far, we have considered similarity measures that only apply to continuous attributes
- Do not confuse discrete/continuous attributes with classification/regression!
- Many datasets have attributes that have a finite number of discrete values (e.g. Yes/No or True/False, survey responses, ratings)
- Once approach to handling discrete attributes is the **Hamming distance**
- Hamming distance is calculated 0 for each attribute where both cases have the same value, 1 for each where they are different
- E.g. Hamming distance between “Stephen" and “Stefann" is 3.

Similarity for discrete attributes

Similarity measures for binary (e.g. yes/no) attributes include:

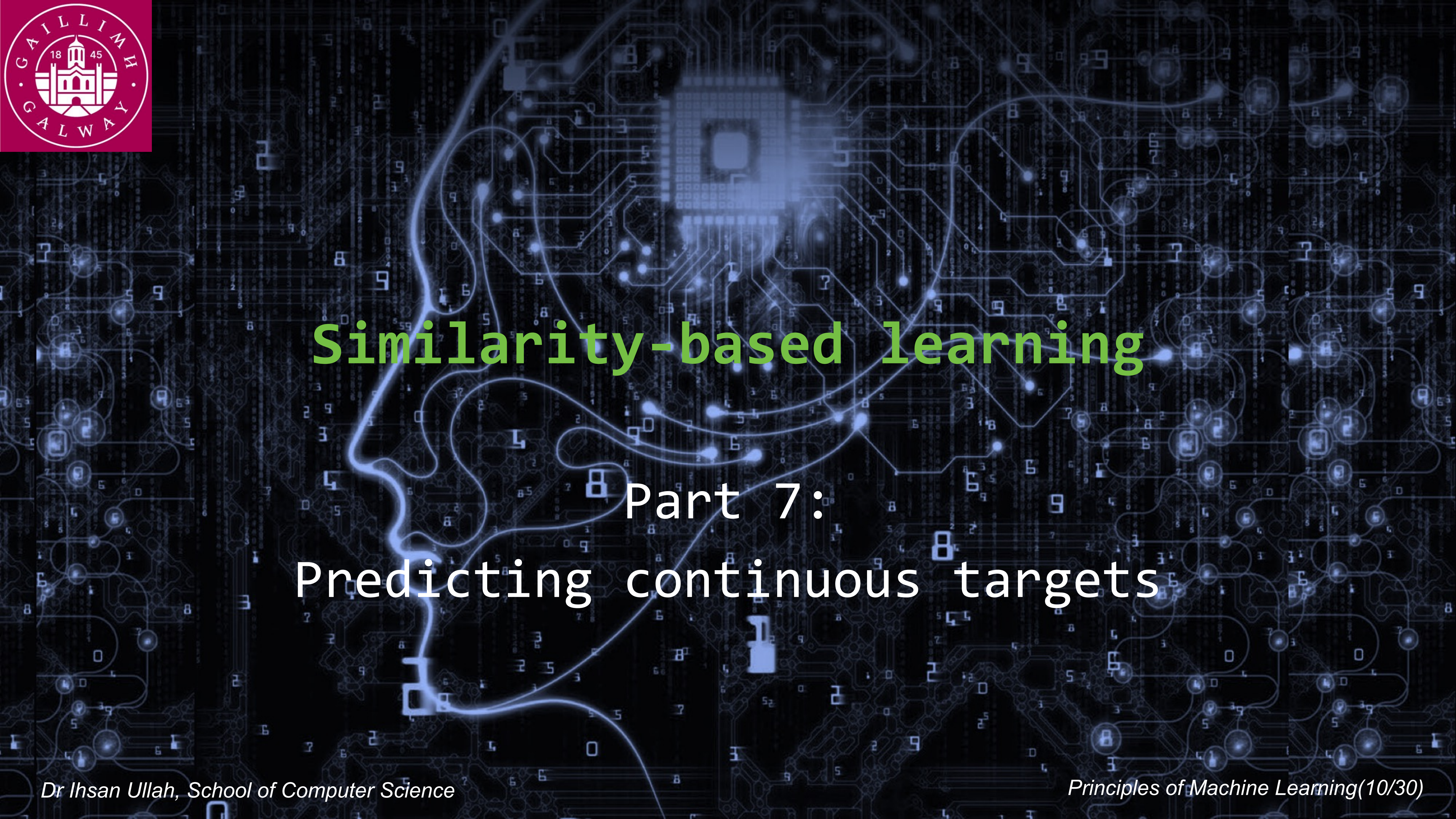
- The **Russel-Rao** similarity index
  - Focuses on measuring “co-presences”, e.g. when comparing users of a web store, compare which pages/products they have clicked on (the attributes are for each product/page, whether that product/page was visited before)
  - The similarity score of two users is based on how many co-presences they share
- The **Sokal-Michener** similarity index
  - Measures similarity using co-presences as well as co-absences
  - Co-absences of attributes may be just as important as co-presences in some domains, e.g. in medical applications, it may be important to compare which symptoms each patient has, as well as which symptoms each patient does not have

Choosing a Distance Metric (1)

- Choices will be limited by attribute data type, e.g. continuous or discrete
- In general, want a metric that:
  - Reflects differences between cases that are important
  - Downplays differences that are irrelevant
  - Application-dependent: work needed here
- E.g. Cosine similarity based on angle, not absolute value
  - May be good to compare objects if we care that they are the same shape but not the same scale
  - Less good if the scale is important

Choosing a Distance Metric (2)

- Possible to design distance metrics of our own
  - Given our understanding of what is important in a domain, can formulate a metric that emphasises what is important and de-emphasises what is not
- Remember that distance metrics must obey **Metric Axioms**:
  1. **Non-negativity**: $metric(\boldsymbol{a}, \boldsymbol{b}) \geq 0$
  2. **Identity**: $metric(\boldsymbol{a}, \boldsymbol{b}) = 0 \iff \boldsymbol{a} = \boldsymbol{b}$
  3. **Symmetry**: $metric(\boldsymbol{a}, \boldsymbol{b}) = metric(\boldsymbol{b}, \boldsymbol{a})$
  4. **Triangular Inequality**: $metric(\boldsymbol{a}, \boldsymbol{b}) \leq metric(\boldsymbol{a}, \boldsymbol{c}) + metric(\boldsymbol{b}, \boldsymbol{c})$
- These axioms may occasionally be violated in similarity measures, but only with good reason!
  - E.g. “A contains B” is non-symmetric



Similarity-based learning

Part 7: Predicting continuous targets

kNN - recap

- *k*-Nearest Neighbour algorithm:
  - Base prediction on several (*k*) nearest neighbours
  - Compute distance from query case to all stored cases, and pick the nearest *k* neighbours

- Classification with kNN:
  - Neighbours vote on classification of test case
- **Regression:**
  - **Average the value of the neighbours**

kNN regression – uniform weighting

$$prediction(\boldsymbol{q}) = \frac{1}{k}\sum_{i=1}^{k} t_i$$

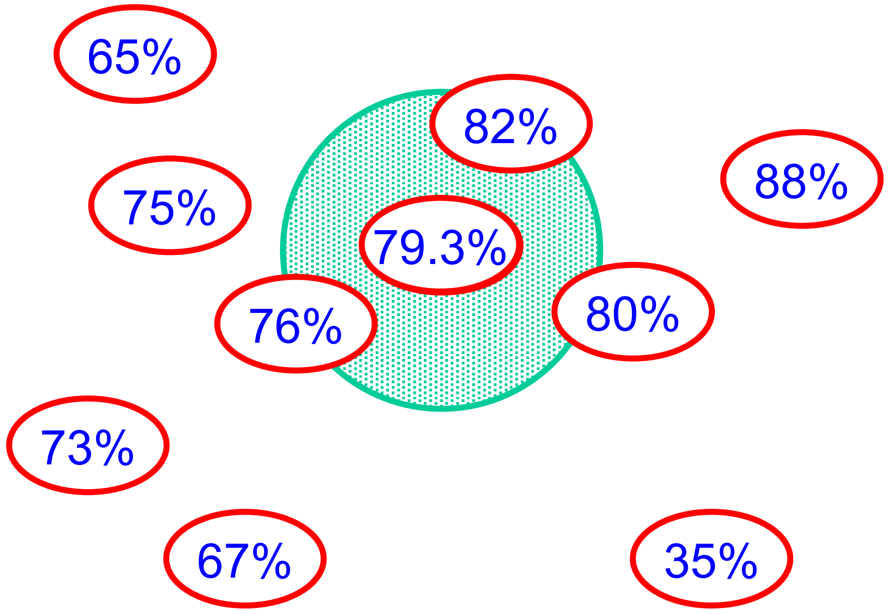
- $\boldsymbol{q}$ is a vector containing the attribute values for the query instance
- *k* is the number of neighbours as before
- t_i is the target value for neighbour *i*
- This assumes that each neighbour is given equal weighting

kNN regression – distance weighting

$$prediction(\boldsymbol{q}) = \frac{\sum_{i=1}^{k}\left(\frac{1}{dist(\boldsymbol{q},\boldsymbol{d}_i)^2}\times t_i\right)}{\sum_{i=1}^{k}\left(\frac{1}{dist(\boldsymbol{q},\boldsymbol{d}_i)^2}\right)}$$

- $\boldsymbol{q}$ is a vector containing the attribute values for the query instance
- $dist(\boldsymbol{q},\boldsymbol{d}_i)$ returns the distance between the query and neighbour *i*
- This assumes that each neighbour is given a weighting based on the inverse square of its distance from the query

k Nearest Neighbours: Visualisation of Regression Example

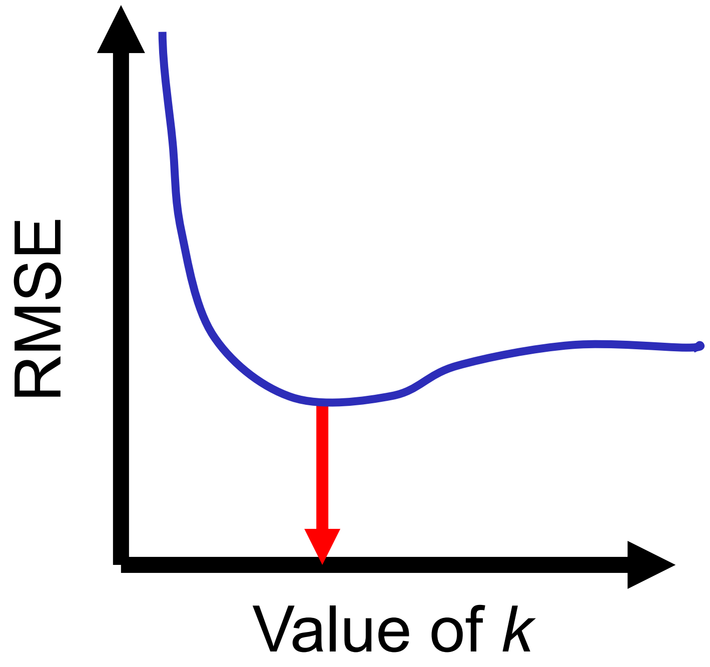


Estimate: 79.3%

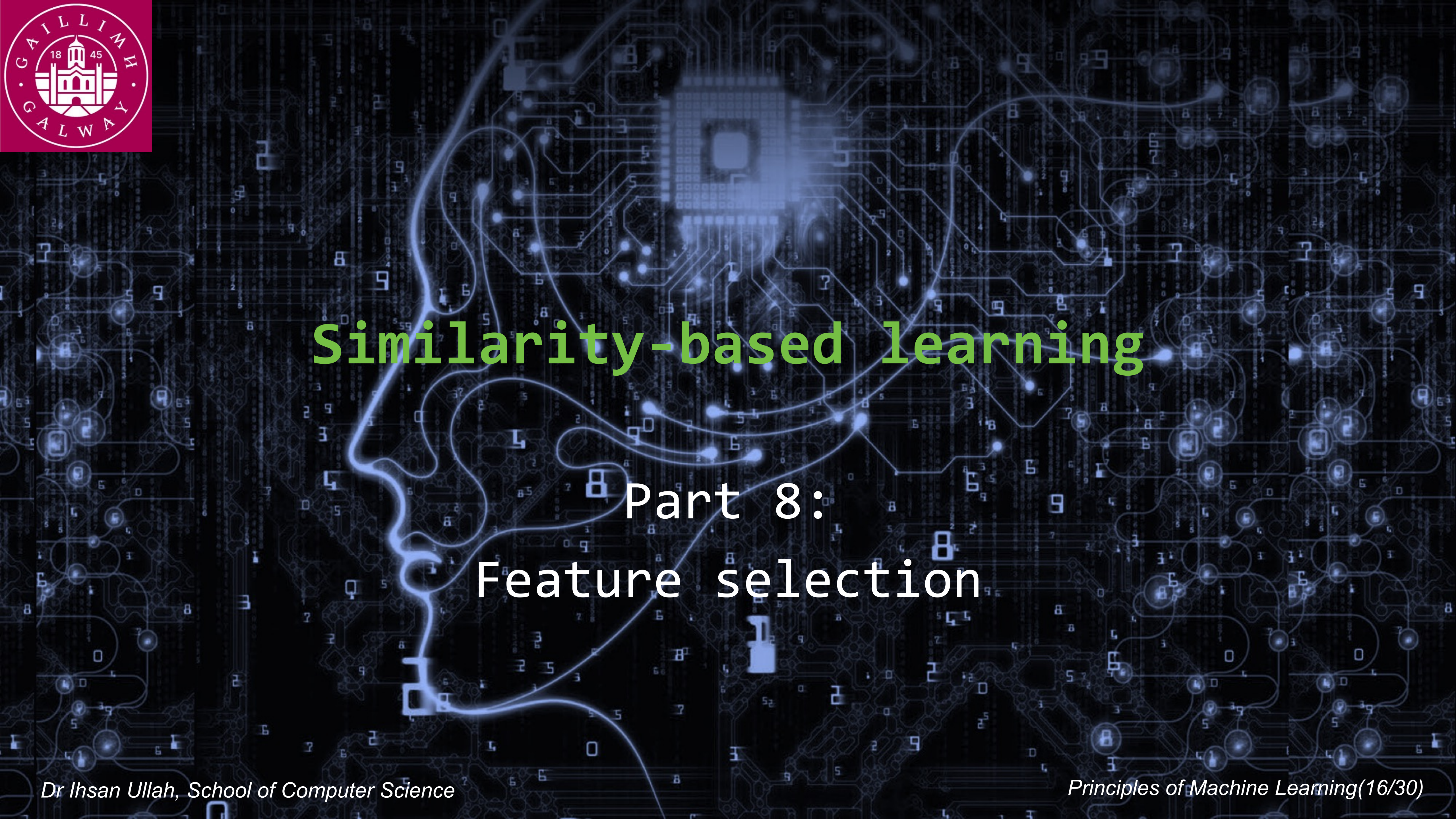
This visualisation assumes Euclidean distance and uniform weighting

Selecting *k* for regression tasks

- As with *k*NN for classification, selecting an appropriate value of *k* is important for regression tasks. How should we set it?
- One solution: test a range of values of *k* and select the one that gives the best score on your chosen evaluation metric, e.g. Root Mean Square Error (RMSE). Should incorporate cross validation when computing the RMSE for each value of *k*
- Note: even values of *k* aren't a problem for regression tasks as majority voting isn't used



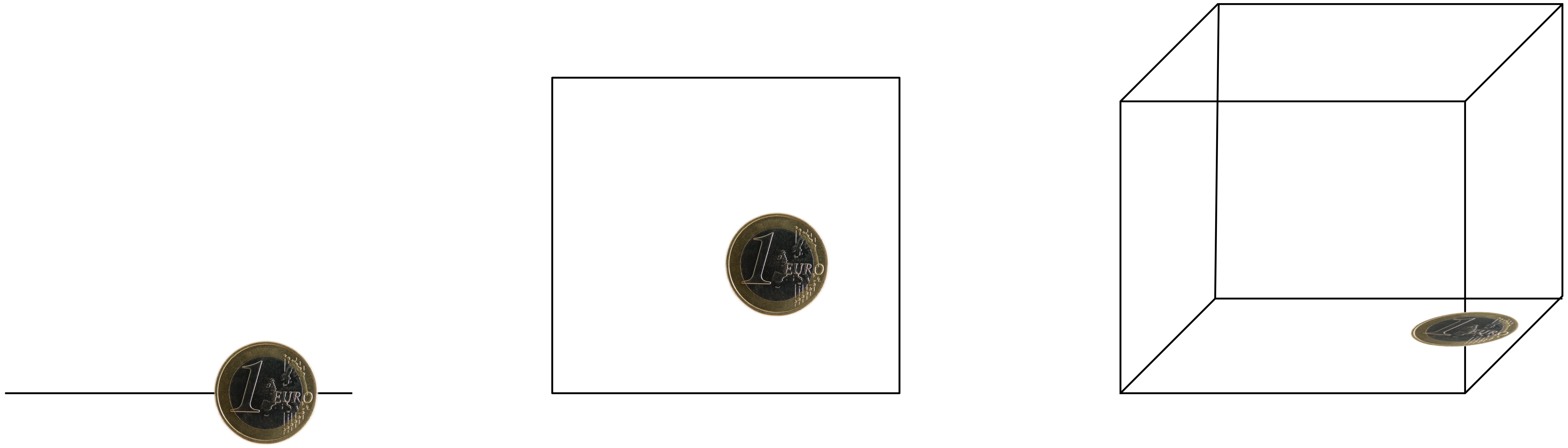
Pick the value of k that Gives the lowest RMSE



Similarity-based learning

Part 8: Feature selection

The Curse of Dimensionality [1]



The Curse of Dimensionality [2]

- Problem — Curse of Dimensionality:
  - Some attributes are much more significant than others
  - All considered equally in distance metric => bad predictions
  - With many attributes, everything becomes 'distant' [see next]
- Solution a:
  - Assign weighting to each dimension (**not** same as distance-weighted kNN!)
  - Optimise weighting to minimise error
- Solution b:
  - Give some dimensions 0 weight: **Feature Subset Selection**

Any algorithm that considers all attributes in a high-dimensional space equally has this problem, not just kNN + Euclidean Distance!

The Curse of Dimensionality [3]

- Russell & Norvig:

  Consider N cases with d dimensions, in hypercube of **unit** volume

  Assume neighbourhoods are hypercubes, length b: volume is b^d

  To contain k points, average neighbourhood must occupy k/N of entire volume

  => $b^d = k/N$

  => $b = (k/N)^{1/d}$

  High dimensions:

  k =10; N =1,000,000; d =100 => b = 0.89
  i.e. neighbourhood spans nearly 90% of each dimension of space!

  Low dimensions:

  k and N unchanged; d =2 => b = 0.003 [OK]

- High-D spaces are generally very sparse: all neighbours far away

Feature Selection

- Fortunately, some algorithms partially mitigate the effects of the curse of dimensionality (e.g. decision tree learning). This is not true for all algorithms however, and heuristics for search can sometimes be misleading!
- kNN and many other algorithms use all attributes when making a prediction
- Acquiring more data is not (always) a realistic option
- The best way to avoid the curse is to use only the most useful features during learning, this process is known as feature selection

Types of features

We may wish to distinguish between different types of descriptive features:

- **Predictive:** provides information that is useful when estimating the correct target value
- **Interacting:** provides useful information only when considered in conjunction with other features
- **Redundant:** features that have a strong correlation with another feature
- **Irrelevant:** doesn’t provide any useful information for estimating the target value

Ideally, a good feature selection approach should identify the smallest subset of features that maintain prediction performance

Feature Selection Approaches

- **Rank and prune:**
  - rank features according to their predictive power and keep only the top X%
  - A **filter** is a measure of predictive power used during ranking, e.g. information gain
  - Drawback: features evaluated in isolation, so we will miss useful interacting features
- **Search for useful feature subsets:**
  - We can pick out useful interacting features by evaluating feature subsets
  - Could generate, evaluate and rank all possible feature subsets then pick best (essentially a brute force approach, computationally expensive/infeasible?)
  - Better approach: **greedy local search**, build feature subset iteratively by starting out with an empty selection, then trying to add additional features incrementally. Requires evaluation experiments along the way. Stop trying to add more features to the selection once termination conditions are met.

Similarity-based learning

Part 9:

Similarity-based learning considerations

Lazy vs Eager Learning [1]

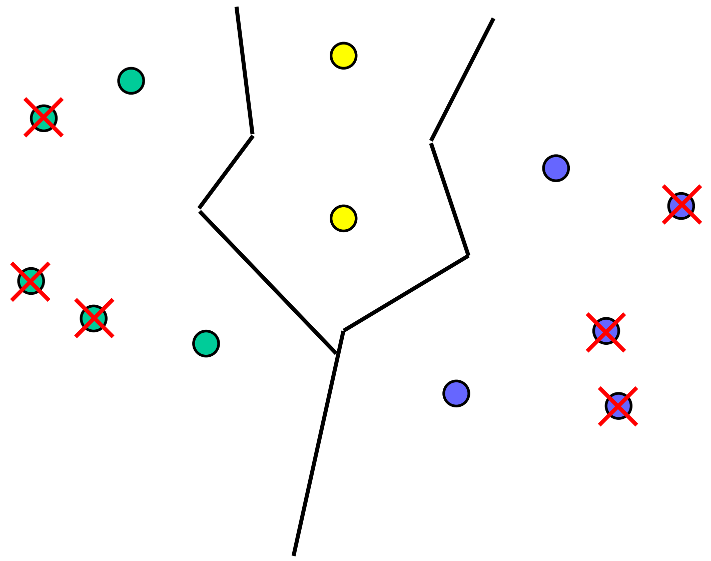
- Eager Learning
  - When given training data, construct model for future use in prediction that summarises the data
  - Analogy: compilation in programming language
  - Slow in model construction,
    quicker in subsequent use
  - Model itself may be useful/informative
- Eager Learning Algorithms:
  - Decision Tree, Artificial Neural Network, …
- On the other hand …

Lazy vs Eager Learning [2]

- Lazy Learning
  - No explicit global model constructed
  - Calculations deferred until new case to be classified
- Creates many local approximations
  - Whereas eager learner must create a global approximation
  - For same form of hypothesis, lazy learner can represent more complex functions
  - E.g. fitting a line to neighbours rather than fitting a line to all of the data
- Lazy Learning Algorithms:
  - 1NN, k Nearest Neighbours, CBR, …
- Concept drift

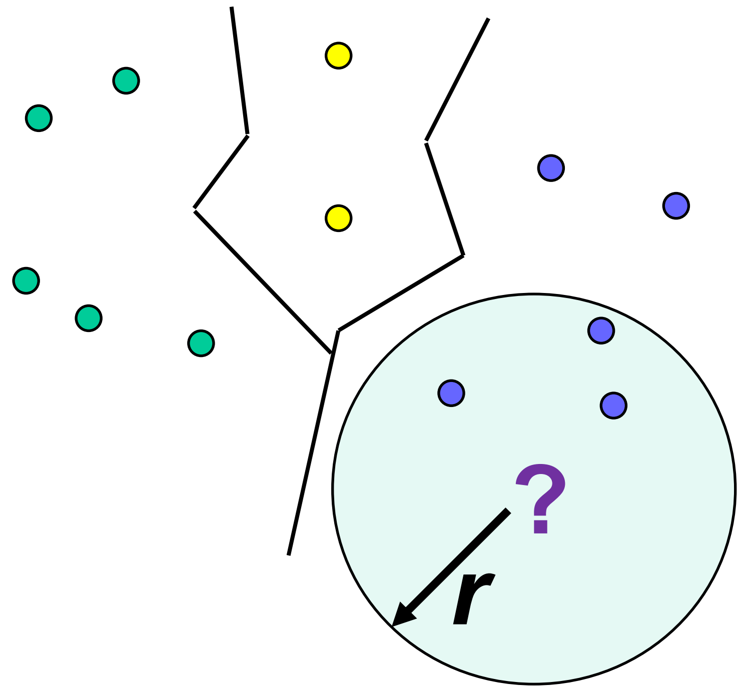
Other Nearest Neighbour variants [1]

- Condensed kNN
  - Classification time proportional to:
    - number of cases
    - number of attributes per case
  - To speed up calculation, eliminate unnecessary ones: Away from decision boundary



Other Nearest Neighbour variants [2]

- Radius-based Nearest Neighbours
  - Instead of selecting *k* nearest neighbours, could also select all neighbours within a certain radius *r*
  - e.g. RadiusNeighborsClassifier in scikit-learn
  - *r* should be chosen carefully
  - Normalisation is very important!
  - Can also specify a default class for instances that have no neighbours within *r* – potentially useful for flagging outliers



When to use kNN

- Consider using when:
  - Plenty of training cases
  - Moderate number of attributes per case (e.g. < 20)
- Benefits:
  - Easy to implement, few parameters to tune (*k* + similarity measure)
  - Comprehensibility: easy to justify decisions
  - Complex target functions => good for **non-coherent concepts** (e.g. “toxic/non-toxic”). Classes don’t have to be linearly separable.
  - No summarisation => information not lost
- Drawbacks:
  - Problems with irrelevant attributes, many attributes
  - May be slow in classification/regression (no summarisation)

Similarity-based learning

Part 10:
Review of topic

Learning Objectives: Review

After completing this successfully, you are now able to …

- Explain what instance-based learning is
- Distinguish between *lazy* and *eager* learning
- Describe operation of k-Nearest Neighbours for classification and regression
- Discuss implications of the *curse of dimensionality*
- Discuss implications of selecting different distance metrics
- Identify suitable applications for kNN and explain how it could be applied