COMP4388: **MACHINE LEARNING**

k-Nearest Neighbour

Dr. Radi Jarrar
Department of Computer Science
Birzeit University

Nearest Neighbour

- One of the earliest classification solutions
- Can be seen as the easiest to implement
- It is a non-parametric instance-based lazy supervised learning technique
- In Nearest Neighbour, the classification decision, of new unlabeled instances, is made based on the data itself

Nearest Neighbour (2)

- No learning stage
- k-Nearest Neighbour classifies a new unlabelled input feature vector ***f*** to the most common class in the set of ***k*** training feature vectors that are the nearest to ***f***
- This classification technique relies on **distance estimation** of the nearest neighbours

Nearest Neighbour (3)

- It uses the training data as templates for classification
- Hence, the name instance-based learning or lazy learning

Applications

- Though this is a simple idea for a classification algorithm, kNN has shown to be a powerful classification technique
- Computer Vision applications: OCR, facial recognition, image-scene classification,…
- Reccommendation system (check out Netflix challenge)
- Pattern recognition (in genetic data to specific deseases for instance)

When it is best used?

- kNN classifier is well-suited for problems in which the relationship between the features and the target classes are complicated, numerous, or very difficult to understand (BUT, instances of the same class tend to look homogeneous)
- If there is no clear distinction among the groups (i.e., classes) then kNN is not really suited for such problems

How it works?

- For a given new input feature vector f^{new}, the nearest neighbour classifier searches for the nearest template in the feature space that minimises the distance to the new input vector as

$$y^* = \arg\min_j d\left(f^{new}, f_j^{train}\right)$$

where f^{new} is the new input feature vector, f_j^{train} is a feature vector from the training dataset, y* is the class to be assigned to f^{new}

How it works? (2)

- The distance d(,) is computed by similarity measures such as Chi^2, Minkowski derivatives (e.g., Euclidean distance, Manhattan distance, …), cosine distance, and other similarity measures
- The Minkowski distance between two vectors $f = (x_1,x_2,...,x_n) \in R^n$ and $q = (y_1,y_2,...,y_n) \in R^n$ is defined as

$$d = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

How it works? (3)

- The previous representation is the simplest representation of a kNN in which a new unlabelled input feature is assigned to the nearest template from the training data
- A more complicated form considers the nearest k nearest neighbours:
  - For a new input vector, the k-Nearest neighbour classifier searches for k templates in the training instances and classifies the input vector into the class with the maximum number of occurrences

Choosing appropriate k

- Deciding how many neighbours to use (i.e., the value of k), determines how well the model will generalise on new data
- Choosing a large value of k reduces the impact caused by noisy data
- Choosing a small value of k allows noisy data (or outliers) to influence the classification of new examples

Choosing appropriate k (2)

- For example, consider setting the value of k equal to n (where n is the number of training feature vectors)
- In this case, every training example is represented in the final vote, the class with the highest number of training examples has majority of voters
- In other words, the model will always predict a new observation as of the most common class regardless of which neighbours are nearest

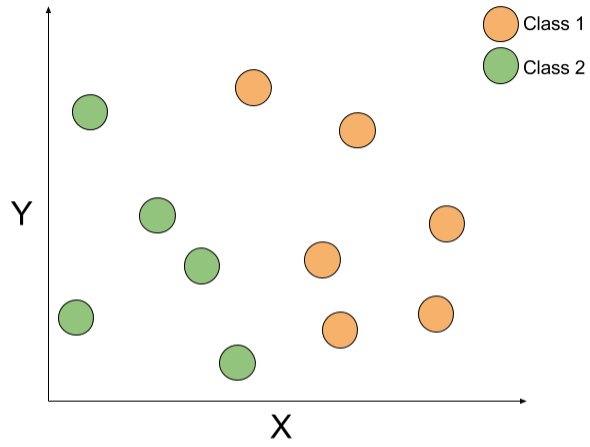
Choosing appropriate k (3)

- Choosing a value for k depends on number of records in the training dataset as well on how hard for the concept to be learned
- Typically, k is set somewhere between 3 and 10
- A common practice is to set the value of k as the square root of n (the size of n)

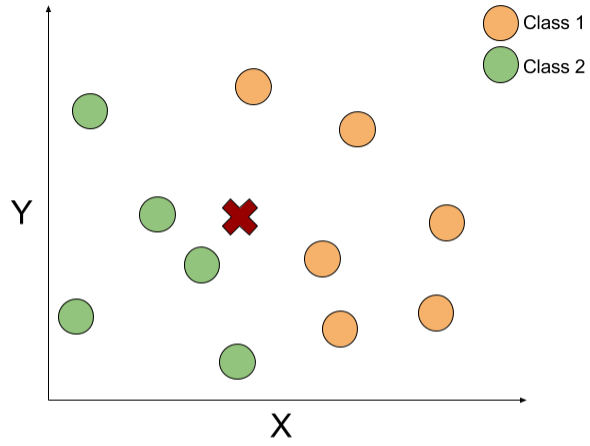
Choosing appropriate k (4)

- Those rules will not always result in a single best value of *k*
- Alternative approach is to test several *k* values and then select the one that results on the best classification performance
- If the size of the dataset is large and the data is not 'very' noisy, the value of *k* is less important

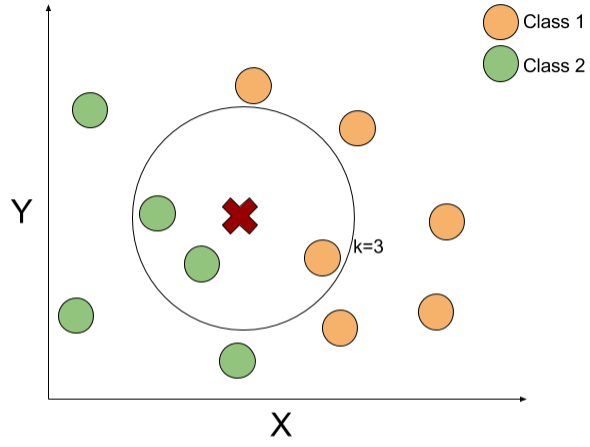
Visualised example



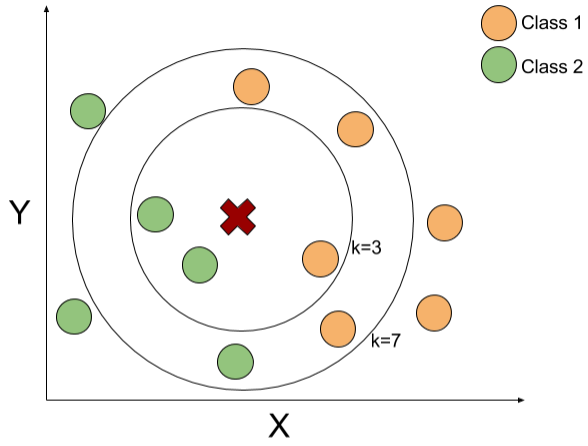
Visualised example



Visualised example – k=3



Visualised example – k=7



Issues in using kNN

- The distance function depends on how features are measured
- This means that if the features are measured on different scales (e.g., one feature has the range 1-10 while another one has a range of 1,000-10,000), then the distance measurements will be dominated by the larger values

Issues in using kNN (2)

- A solution to this problem is to scale all features such that they contribute equally to the distance formula
- Scaling features means to make all features on the same scale. E.g., if one feature is on the scale 1-10 and another 1,000-10,000; then scaling the features will result in having the second feature on the scale of 1-10 as well

Nominal features

- Minkowski distance formula (and its derivatives) is not suited to deal with nominal features
- In such case, the nominal features should be converted to a numeric format
- Dummy coding technique can be utilised such that a value of 1 indicates a category and a value of 0 indicates the others

Nominal features (2)

- Back to the 'PlayTennis' dataset, the feature 'Wind' can be converted as follows: strong=1 and 0 otherwise (which is only weak)
- Consider the case of more than 2 possibilities; i.e., 'Outlook' how to represent it?
- Hint: create new columns!

Nominal features (3)

- This is achieved as follows: a new column is created having the value 1 if 'Outlook' is 'Sunny' and 0 otherwise
- This means this column will have a value of the outlook is Sunny
- Another column is created having the value 1 if 'Outlook' is 'Overcast' and 0 otherwise
- This means if 'Outlook' is 'Rain', then the values of both new columns are zero and zero
- For n-category nominal feature, it can be dummy coded by creating binary indicator variables for n-1 levels of the feature

Strength of kNN

- Simple yet effective
- No assumptions about the underlying training data is made
- Fast training phase

Weaknesses of kNN

- Slow classification phase
- Requires high memory in case of large data
- Nominal and missing features require extra processing
- No model is produced: every time a new unlabelled instance is given, the code will be run over the training instances
- Having no model limits the ability to find relationships among the features