UNIT 2 U2/0/2024

Machine Learning

COS4852

Year module

Department of Computer Science

School of Computing

CONTENTS

This document contains the material for UNIT 2 for COS4852 for 2024.

Instance-based learning, or sometimes called memory-based learning, is a group of learning algorithms that does not perform explicit generalization. It rather uses a process of compare new problem instances with instances seen in training, which have been stored in memory (hence memory-based). Computation is postponed until a new instance is observed. These algorithms are therefore referred to as “lazy”.

The most common instance-based learners are the family of *k*-NEAREST NEIGHBOURS (*k*NN) algorithms, Kernel-based networks (of which the Support Vector Machine (SVM) is the most well known), and Radial Basis Function (RBF) networks. The last two classes of learners belong with Neural Network. In this unit we will therefore only focus on *k*NN as an example of this type of learning.

1 OUTCOMES

In this unit you will describe and solve a learning problem using techniques of Instance-based Learning. You will learn more about the theoretical basis of one kind of instance-based learner, namely *k*-NEAREST NEIGHBOURS (*k*NN), and understand how to apply the algorithm. You will also study locally weighted regression and the radial basis function network, as further examples of Instance-based Learning. You will:

1. Learn about the theoretical basis of the nearest neighbours approach.
2. Understand what kinds of problems can be solved using *k*NN.
3. Understand how the *k*NN algorithm works.
4. Learn how to solve classification problems using *k*NN.
5. Learn how the locally weighted regression works.
6. Learn how the radial-basis function networks works.

After completion of this Unit you will be able to:

1. Design a *Classification System* using *k*NN.
2. Discuss the theoretical basis of *k*NN and its variants.
3. Understand the advantages and limitations of *k*NN.
4. Discuss what kinds of problems can be solved using *k*NN.
5. Solve classification problems by implementing the *k*NN algorithm on given data sets.

2 INTRODUCTION

In this Unit you will investigate the theory behind nearest neighbours and learn to implement the *k*NN algorithm.

One thing we want a Machine Learning algorithm to do, is to *generalise* – this refers to the model's ability to perform well on new unseen data rather than just the data that it was trained on. Machine Learning models are most often used to make predictions on new unknown data. It is not sufficient to have a good performance on the training data. The real test is when the model perform well on new instances.

There are two main approaches to generalization: model-based learning and instance-based learning.

Model-based learning have a set of parameters (sometimes these are huge) that do not change as the size of the training data changes. These models are referred to as *parametric*.

Instance-based learning is sometime also referred to as memory-based learning. Such models do not perform any kind of generalisation, but rather compares new instance with those in memory, using some similarity measure.

*k*NN is the simplest of these algorithms, and works on the principle that similar instances tend to form clusters.

3 PREPARATION

3.1 Textbooks

Chapter 8 of Tom Mitchell's book gives a detailed working of the *k*NN algorithm and some of its derivatives.

Section 2.5.2 of Bishop's book gives a kernel-based approach to *k*NN.

Chapter 2 of Harrington's book gives a detailed working of the basic *k*NN algorithm using Python, including all the detailed steps such as data preparation and visualisation.

3.2 Online material

The original ideas around *k*NN were created by Fix and Hodges in their 1951 paper. Cover and Hart improved on the idea in 1967 with their paper "Nearest Neighbor Pattern Classification.".

The IBM site provides a good overview of the *k*NN algorithm, how to use different distance metrics, and a brief, but incomplete bit on choosing *k*.

This Tutorialspoint link gives a step-by-step description of the basic kNN algorithm, some Python code to implement it, as well as code for using kNN as a regressor.

These course notes discuss weighted kNN and some others.

An excellent, detailed discussion on kNN are in these lecture notes from UWM.

4 DISCUSSION

A note on k on websites: When you look at some of the online resources you will sometimes see that they refer to K and k. In CS (and mathematics) notation is very important, and there is a distinct difference between capitals and not. in kNN we always refer to k, since this a variable, and a critical one for the success or not of a particular run of the algorithm.

4.1 The kNN algorithm

The k-NEAREST NEIGHBOURS (kNN) algorithm is one of the simplest, though widely useful, classification algorithms. It works on the principle that instances of the same class tend to cluster together. In other words, a new instance is very likely to be of the same class as those closest to it.

A target function $f : X \rightarrow Y$, is represented by a set of n instances $\langle X_i, Y_i \rangle$, where $X = \{X_1, X_2, \ldots, X_n\}$ are a set of attribute values. These attribute values could be coordinates, or any combination of values that belong to a specific instance. Y_i typically represent a single class value that matches the attribute values of X_i. When a new instance $X_j = \{X_{j1}, X_{j2}, \ldots, X_{jn}\}$, of unknown class has to be classified, kNN calculates the distance between X_j and each of the other instances. The k nearest neighbours are selected, and their class values counted to determine the majority class. This majority class is then assigned to the new instance X_j.

The distance measure is selected to match the data types of the instance attributes. These include the Euclidean distance, and the Manhattan distance. There are several others that are used. For example, if the attributes are coordinate values, the Euclidean distance measure works well.

4.2 *1*-NEAREST NEIGHBOURS

To get a grasp of how the algorithm works, we can reduce kNN to its simplest case, where $k = 1$, i.e. *1*-NEAREST NEIGHBOURS. Any new instance will be classified as the same cass as its nearest neighbour. Figure 1 shows the neighbourhoods (in blue lines) surrounding each instance. Any new instance that falls within any of these blue bounding boxes will be classified as the class of the instance at its center. For example, a new instance at $new_1 \leftarrow (5, 3)$ will be classified the same as N_5 (a red triangle), while a new instance at $new_2 \leftarrow (4, 3)$ will be classified the same as P_4 (a green circle).

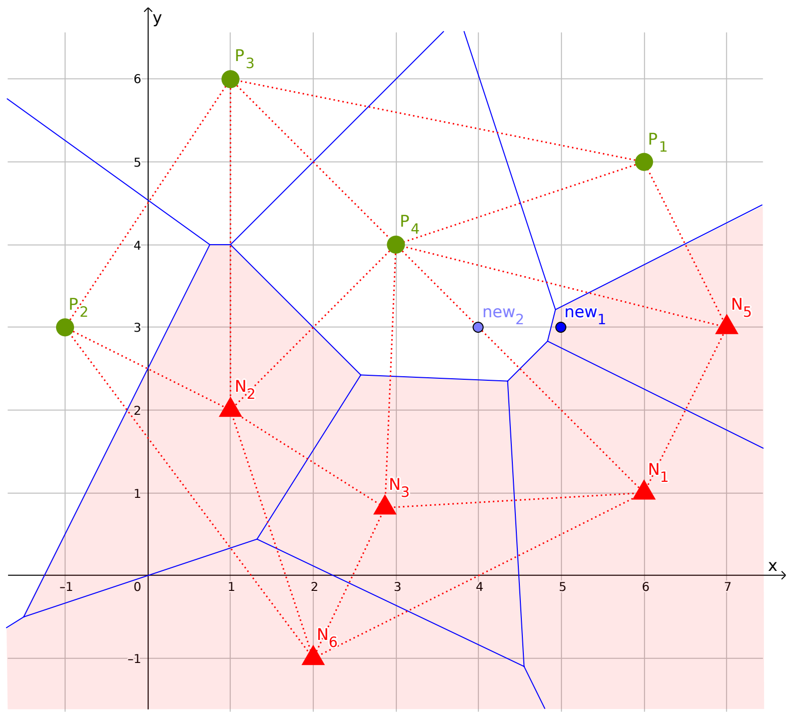

Figure 1: The case of $k = 1$: *1*-NEAREST NEIGHBOURS.

4.3 Finding a good value for k

The value of k is critical to the algorithm. With $k = 1$ we have *1*-NEAREST NEIGHBOURS, and the new instance will be assigned the class of the nearest neighbour, which may be an outlier, and therefore not be an accurate representation of the classes. Small values of k may lead to over-fitting. Larger values of k can lead to under-fitting. If $k = n$, the class value of all the instances are used to find a majority - there is no point in calculating the distances. Clearly there are values of k that are close to optimal. A simple heuristic value, that is often used is $k = \sqrt{n}$, or more specifically the nearest uneven integer to $\sqrt{n}$. If there are an even number of classes, such as a binary classification, the value of k should be uneven so that there is always a majority outcome. A more accurate, but time-consuming approach would be to use cross-validation to test a range of values of k, but this comes at a computation price. An n-fold cross-validation increases the computation n times.

4.4 *k*NN worked example

An example will illustrate the workings of the algorithm. Consider the instance set in Figure 2, showing 8 instances of two classes A and B. A new instance P_9 at $(2, 1)$ has an unknown class.

Use kNN to determine the new class for C. Using the heuristic k should be chosen as $k = 3$, but to illustrate the effect of different distance measures, use $k = 5$. In other words find the 5 nearest neighbours to C.

Use the Euclidian distance measure

$$d_{Euclidian}(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \tag{1}$$

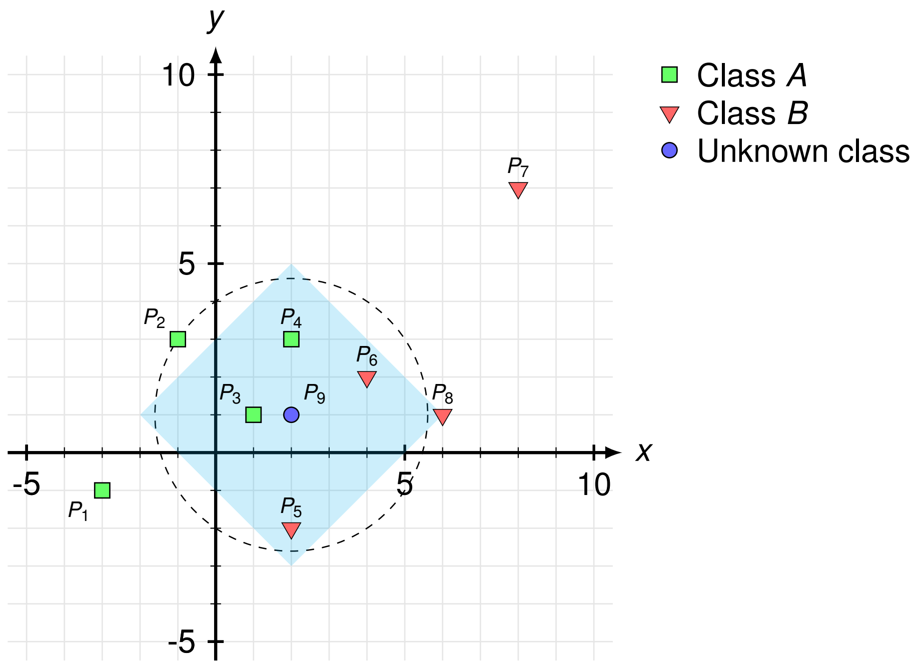

Figure 2: Example instance space for kNN with two classes A and B.

to calculate the distance between P_9 and the other 8 instances, and rank them according to the closest distance.

instance	$d_{Euclidian}(P_9, P_i)$	class	rank
P_1	5.385	A	7
P_2	3.606	A	5
P_3	1.000	A	1
P_4	2.000	A	2
P_5	3.000	B	4
P_6	2.236	B	3
P_7	8.485	B	8
P_8	4.000	B	6

With $k = 5$, the 5 closest neighbours gives 3 instances of class A and 2 instances of class B. The majority is therefore class A, hence P_9 is assigned class A. The dashed circle in Figure 2, with radius $r = 3.606$ shows the minimum Euclidian radius that encloses the 5 closest neighbours to P_9.

Now use the Manhattan distance measure

$$d_{Manhattan}(p, q) = |p_x - q_x| + |p_y - q_y| \tag{2}$$

to do the same calculation (read up on the Hamming and the Cityblock distance measures).

instance	$d_{Euclidian}(P_9, P_i)$	class	rank
P_1	7	A	7
P_2	5	A	6
P_3	1	A	1
P_4	2	A	2
P_5	3	B	3
P_6	3	B	4
P_7	12	B	8
P_8	4	B	5

Now, the 5 closest neighbours gives a different result, with 2 instances of class A and 3 instances of class B. The majority is therefore class B, hence P_9 is assigned class B. The cyan diamond in Figure 2, with Manhattan radius $r = 4$ shows the minimum Manhattan radius that encloses the 5 closest neighbours to P_9. This illustrates the importance of choosing the correct distance measure for the data set. If x and y are simply coordinates, the Euclidian distance measure is appropriate, but if x and y represent natural numbers (say x are the number of petals on a flower, and y is the number of sees lobes), then the Manhattan distance may be a better choice.

It is often a good idea to normalise the data, so that all attributes fall within the same range, i.e. have the same scale so that distance measures compares the attributes equally.

4.5 Multi-valued classification and Regression

The algorithm above looked at a binary classification problem. We have discrete-valued instances, and only two classes. Classification can be extended to multiple classes. Handwritten character recognition is an example where there are at least 36, and with capital letter and special characters and ligatures, far more classes. kNN can be extended in a straightforward manner to handle multi-class data.

This GeeksForGeeks page gives a brief overview of the difference between classification and regression.

Here is another discussion that goes into some depth to explain how kNN can be used when we have real-valued data, i.e. a regression problem and not a classification problem.

4.6 Weighted *k*NN

The majority voting process can at times be a problem. This happens when the distribution of the data is skewed, i.e. there are more instances of one class than the others. This class will dominate any straightforward counting mechanism, such as majority voting. One way to address this problem is to scale the distance values so that instances closer to the new one contribute more to outcome than those further away. This is idea behind the weighted nearest neighbour algorithm.

If we assign a weight values to all instances, the normal *k*NN algorithm can be re-written so that we assign a weight of $1/k$ to each of the k nearest neighbours to the new instance, and a weight of 0 to all the other instances. We can then generalise the *k*NN algorithm to a weighted algorithm.

The standard Euclidian distance between instances $\mathbf{x}^{[a]}$ and $\mathbf{x}^{[b]}$ can be expressed as (this is the same as Eq. 1, expressed in a vector/matrix notation):

$$d\left(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}\right) = \sqrt{\left(\mathbf{x}^{[a]} - \mathbf{x}^{[b]}\right)^T \left(\mathbf{x}^{[a]} - \mathbf{x}^{[b]}\right)} \tag{3}$$

We can now weigh each distance. All these weights can be written in the form of a matrix **W**, and the weighted distance measure becomes:

$$d_w\left(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}\right) = \sqrt{\left(\mathbf{x}^{[a]} - \mathbf{x}^{[b]}\right)^T \mathbf{W} \left(\mathbf{x}^{[a]} - \mathbf{x}^{[b]}\right)} \tag{4}$$

All this means in practice is that we multiply each distance by a weight value. These weights can have any value, but for it to work well we design our weight assignments to increase the effect of closer neighbours and decrease the effect of further neighbours. Tom Mitchell gives the example of the inverse squared distance, so that the weight for the distance between instance i and t becomes:

$$w^{[i]} = \frac{1}{d\left(\mathbf{x}^{[i]}, \mathbf{x}^{[t]}\right)^2} \tag{5}$$

If all of the distance weights have a non-zero value (as it would in Eq. 5), the algorithm becomes *global*, so instead of only considering the nearest k values, all instances contribute to the new classification.

4.7 Pros and Cons of *k*NN

Advantages: The algorithm does not need to build a model of the entire data set. It makes local approximations to the target function. New data can be added, and the algorithm adapts to new data.

Disadvantages: Classification costs are high, because the algorithm needs to use all the data to incorporate a new instance, or classify an unknown instance. All die data needs to be kept in memory (RAM or disk) required to store the data. Each query involves builds the (local) model from scratch.

Section 2.13 of the UWM course discusses more on the advantages and disadvantages of *k*NN.

5 Locally Weighted Regression

Locally weighted linear regression is another example of instance-based learning. It does not learn a fixed set of parameters (in contrast to ordinary linear regression), but parameters θ are computed individually for each query point x. When computing θ the algorithms gives higher preference to instances closer to the vicinity of x than the points lying further away. There is also no training phase – all the calculations are done at the time when a prediction gets made. In this sense it is similar to kNN.

This CodeStudio site gives a good overview od the algorithm as well as Python code to illustrate its implementation.

A similar exercise is done in this GeeksforGeeks page.

A slightly more theoretical approach is given in this Medium page.

6 Radial-basis function networks

In the Unit on Neural Networks you will learn how a network of neurons are constructed, and how such a network can be trained. The classic neural network uses activation functions that approximate a global function for the data set being trained. A radial-basis network uses a similar construction, but uses local probability density functions in each neuron to create local linear approximations, that get combined to create a global approximation. In that sense this is an example of instance-based learning. For now you need to understand what kernel function is, and you ony need to look at the common Gaussian kernel function (which uses the well-known Gaussian disribution function).

As you deduce from the short introduction above, the RBF network does not completely fall in the pure instance-based class, as it has a training phase. It can be thought of as a simplified neural network. Once you've done the Unit on Neural Networks you should understand its place better.

This Towards Data Science page gives a good overview of the RBF network, how to train it, and gives an examples of its application to the XOR function.

7 ACTIVITIES

What do we mean when we say that kNN is a "lazy" algorithm? Discuss this in detail.

Find material on more advanced variations of kNN and discuss in detail.

Find material on Kernel-based learning algorithms. Discuss in detail their the relationship with kNN. Discuss Radial-Basis Functions as Machine Learning algorithms in detail.

Find more material on Support Vector Machines (SVMs). Discuss the workings of these algorithms in detail, and their relationship to kNN.

Find more material on Radial Basis Function (RBF) networks. Discuss the workings of these algorithms in detail. Discuss their relationship to logistic regression.
