School of Computing and Information Systems
the University of Melbourne
COMP90049 Introduction to Machine Learning (Semester 1, 2023)

Week 4: Sample Solutions

1. What is optimisation? What is a "loss function"?

   Optimization is the process of finding the best solution or set of solutions to a problem that maximizes or minimizes a given objective function while satisfying certain constraints. In machine learning, optimization is the process of finding the best set of parameters (**optimal parameters**) for a model that minimizes the error or loss on a **training** set of data.

   In machine learning, a loss function, also known as a cost function or objective function, is a function that measures the difference between the predicted output of a model and the true output. The loss function is used to guide the optimization process by providing a measure of how well the model is performing on the **training** data. The goal of optimization is to minimize the value of the loss function by adjusting the model parameters. We want to minimise undesirable outcomes (errors). To do so, we define a function that best describes our *undesirable outcomes* for each model. This function is called a *cost function* or a *loss function*.

   The choice of a loss function depends on the specific problem and the desired performance metric. A good loss function should be differentiable and continuous, so that it can be optimized using gradient descent or other optimization algorithms. It should also be able to capture the specific characteristics of the problem and the desired performance metric.

2. For the following dataset:

	apple	*ibm*	*lemon*	*sun*	CLASS
TRAINING INSTANCES					
A	4	0	1	1	FRUIT
B	5	0	5	2	FRUIT
C	2	5	0	0	COMPUTER
D	1	2	1	7	COMPUTER
TEST INSTANCES					
T1	2	0	3	1	?
T2	1	2	1	0	?

   A. Using the **Euclidean distance** measure, classify the test instances using the 1-NN method.

      In this method we need to calculate the distance between a test instance and a prototype. To do so we need a to use a similarity/distance function. Here our distance function is the Euclidian Distance:

$$d_E(A, B) = \sqrt{\sum_k (a_k - b_k)^2}$$

      Using this function, we will calculate the distance between the test instance and each training instance:

$$d_E(T_1, A) = \sqrt{(2-4)^2 + (0-0)^2 + (3-1)^2 + (1-1)^2} = \sqrt{8} \approx 2.828$$

$$d_E(T_1, B) = \sqrt{(2-5)^2 + (0-0)^2 + (3-5)^2 + (1-2)^2} = \sqrt{14} \approx 3.742$$

$$d_E(T_1, C) = \sqrt{(2-2)^2 + (0-5)^2 + (3-0)^2 + (1-0)^2} = \sqrt{35} \approx 5.916$$

$$d_E(T_1, D) = \sqrt{(2-1)^2 + (0-2)^2 + (3-1)^2 + (1-7)^2} = \sqrt{45} \approx 6.708$$

The nearest neighbour is the one with the smallest distance — here, this is instance `A`, which is a `FRUIT` instance. Therefore, we will classify this instance as `FRUIT`.

The second test instance is similar:

$$d_E(T_2, A) = \sqrt{(1-4)^2 + (2-0)^2 + (1-1)^2 + (0-1)^2} = \sqrt{14} \approx 3.742$$

$$d_E(T_2, B) = \sqrt{(1-5)^2 + (2-0)^2 + (1-5)^2 + (0-2)^2} = \sqrt{40} \approx 6.325$$

$$d_E(T_2, C) = \sqrt{(1-2)^2 + (2-5)^2 + (1-0)^2 + (0-0)^2} = \sqrt{11} \approx 3.317$$

$$d_E(T_2, D) = \sqrt{(1-1)^2 + (2-2)^2 + (1-1)^2 + (0-7)^2} = \sqrt{49} = 7$$

Here, the nearest neighbour is instance `C`, which is a `COMPUTER` instance. Therefore, we will classify this instance as `COMPUTER`.

B. Using the **Manhattan distance** measure, classify the test instances using the 3-NN method, for the three weightings we discussed in the lectures: *majority class*, *inverse distance*, *inverse linear distance*.

The first thing to do is to calculate the Manhattan distances, which is like the Euclidean distance, but without the squares/square root:

$$d_M(A, B) = \sum_k |a_k - b_k|$$

$$d_M(T_1, A) = |2-4| + |0-0| + |3-1| + |1-1| = 4$$

$$d_M(T_1, B) = |2-5| + |0-0| + |3-5| + |1-2| = 6$$

$$d_M(T_1, C) = |2-2| + |0-5| + |3-0| + |1-0| = 9$$

$$d_M(T_1, D) = |2-1| + |0-2| + |3-1| + |1-7| = 11$$

$$d_M(T_2, A) = |1-4| + |2-0| + |1-1| + |0-1| = 6$$

$$d_M(T_2, B) = |1-5| + |2-0| + |1-5| + |0-2| = 12$$

$$d_M(T_2, C) = |1-2| + |2-5| + |1-0| + |0-0| = 5$$

$$d_M(T_2, D) = |1-1| + |2-2| + |1-1| + |0-7| = 7$$

The nearest neighbours for the first test instance are `A`, `B`, and `C`. For the second test instance, they are `C`, `A`, and `D`.

The **majority class** weighting method:

In this method we effectively assign a weight of 1 to every instance in the set of nearest neighbours:

- For the first test instance, there are 2 FRUIT instances and 1 COMPUTER instance. There are more FRUIT than COMPUTER, so we predict FRUIT.

- For the second test instance, there are 2 COMPUTER instances and 1 FRUIT instance. There are more COMPUTER than FRUIT, so we predict COMPUTER.

The **inverse distance** weighting method:

In this method we first need to choose a value for ϵ, let's say 1:

- For the first test instance:
    - The first neighbour (a FRUIT) gets a weight of $\frac{1}{d+\epsilon} = \frac{1}{4+1} = 0.2$
    - The second neighbour (a FRUIT) gets a weight of $\frac{1}{d+\epsilon} = \frac{1}{6+1} \approx 0.14$
    - The first neighbour (a COMPUTER) gets a weight of $\frac{1}{d+\epsilon} = \frac{1}{9+1} = 0.1$

  Overall, FRUIT instances have a score of 0.2+0.14 = 0.34, and COMPUTER instances have a score of 0.1, so we would predict FRUIT for this instance.

- For the second test instance:
    - The first neighbour (a COMPUTER) gets a weight of $\frac{1}{d+\epsilon} = \frac{1}{5+1} \approx 0.17$
    - The second neighbour (a FRUIT) gets a weight of $\frac{1}{d+\epsilon} = \frac{1}{6+1} \approx 0.14$
    - The first neighbour (a COMPUTER) gets a weight of $\frac{1}{d+\epsilon} = \frac{1}{7+1} = 0.12$

  Overall, FRUIT instances have a score 0.14, and COMPUTER instances have a score of 0.17+0.12=0.29, so we would predict COMPUTER for this instance.

  Note: If we have used Euclidean distance (instead of Manhattan distance) would give a different result here.

The **inverse linear distance** weighting method:

In this method we are going to weight instances by re-scaling the distances according to the following formula, where d_j is the distance of the j^{th} nearest neighbour:

$$w_j = \frac{d_3 - d_j}{d_3 - d_1}$$

Note: Compared to the lecture version, we have substituted k = 3 here, because we are using the 3-Nearest Neighbour method.

- For the first test instance:
    - The first neighbour (a FRUIT) gets a weight of $\frac{d_3-d_1}{d_3-d_1} = \frac{9-4}{9-4} = 1$
    - The second neighbour (a FRUIT) gets a weight of $\frac{d_3-d_2}{d_3-d_1} = \frac{9-6}{9-4} = 0.6$
    - The first neighbour (a COMPUTER) gets a weight of $\frac{d_3-d_3}{d_3-d_1} = \frac{9-9}{9-4} = 0$

  Overall, FRUIT instances have a score of 1+0.6 = 1.6, and COMPUTER instances have a score of 0, so we would predict FRUIT for this instance.

- For the second test instance:
    - The first neighbour (a COMPUTER) gets a weight of $\frac{d_3-d_1}{d_3-d_1} = \frac{7-5}{7-5} = 1$

- The second neighbour (a FRUIT) gets a weight of $\frac{d_3-d_2}{d_3-d_1} = \frac{7-6}{7-5} = 0.5$
- The first neighbour (a COMPUTER) gets a weight of $\frac{d_3-d_3}{d_3-d_1} = \frac{7-7}{7-5} = 0$

Overall, FRUIT instances have a score of 0.5, and COMPUTER instances have a score of 1+0=1, so we would predict COMPUTER for this instance.

C. Can we do weighted k-NN using **cosine similarity**?

Of course! Cosine similarity is a similarity measure used to compute the similarity between two vectors in a high-dimensional space. One key difference between cosine similarity and Euclidean or Manhattan distance is that cosine similarity is based on the angle between the vectors, while Euclidean or Manhattan distance is based on the magnitude of the vectors. This means that cosine similarity is not affected by the length of the vectors, only by their orientation or direction, while Euclidean or Manhattan distance takes both magnitude and orientation into account.

Cosine similarity is often preferred over Euclidean or Manhattan distance in text classification tasks, because it can better handle high-dimensional data, such as text data, where the vectors can be very sparse and have many dimensions. This is because cosine similarity is not affected by the magnitude of the vectors, only by their orientation, so it can more accurately capture the similarity between two documents based on their word frequencies.

An overall weighting for a class using Cosine Similarity can be obtained by summing the cosine scores for the instances of the corresponding class, from among the set of nearest neighbours.

Let's summarise all these predictions in a table (overleaf). We can see that there is some divergence for these methods, depending on whether B or D is the 3rd neighbour for T_2:

Inst	Measure	k	Weight	Prediction
T_1	d_E	1	-	FRUIT
		3	Maj	FRUIT
		3	ID	FRUIT
		3	ILD	FRUIT
	d_M	1	-	FRUIT
		3	Maj	FRUIT
		3	ID	FRUIT
		3	ILD	FRUIT
	cos	1	-	FRUIT
		3	Maj	FRUIT
		3	Sum	FRUIT
T_2	d_E	1	-	COMPUTER
		3	Maj	FRUIT
		3	ID	FRUIT
		3	ILD	COMPUTER
	d_M	1	-	COMPUTER
		3	Maj	COMPUTER
		3	ID	COMPUTER
		3	ILD	COMPUTER
	cos	1	-	COMPUTER
		3	Maj	FRUIT
		3	Sum	FRUIT