School of Computing and Information Systems
The University of Melbourne
COMP90049 Introduction to Machine Learning (Semester 2, 2020)

Tutorial exercises: Week 5

1. For the following dataset:

apple	*ibm*	*lemon*	*sun*	CLASS
TRAINING INSTANCES				
4	0	1	1	FRUIT
5	0	5	2	FRUIT
2	5	0	0	COMPUTER
1	2	1	7	COMPUTER
TEST INSTANCES				
2	0	3	1	?
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 calculating 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 assigns 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! If anything, this is easier than with a distance, because we can assign a weighting for each instance using the cosine similarity directly. An overall weighting for a class 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 of 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

2. What is **gradient descent**? Why is it important?

Gradient descent is an instance of iterative optimization algorithms: it finds the parameters corresponding to optimal points of a target function step-by-step, by starting out with some initial parameter value, and incrementally modifying this value in the `most promising way', i.e., in the way that leads to the largest improvement of the target function. This step-by-step procedure is important for optimization problems with no closed-form solution. This has numerous applications - most intuitively, in determining the regression weights which minimise an error function over some training data set.

3. [OPTIONAL] (a) What is **Regression**? How is it similar to **Classification**, and how is it different?

Regression and classification are both categorized under the same umbrella of *supervised* machine learning. Both share the same concept of utilizing known datasets (labelled training datasets) to make predictions.

Our target attribute (class) is *nominal* in classification, but *numeric* (continuous) in Regression. Consequently, in regression we can't exhaustively assess the likelihood of each class.

(b) Come up with one typical classification task, and one typical regression task. Specify the range of valid values of y (results) and possible valid values for x (attributes).

Regression example: Predicting house prices. Possible attributes can be x={location, size, age, interest rates, ...}. *y* is a real value (price) and strictly positive.

Classification example: Sentiment classification of star movie reviews. x={set of words, author ID, review *lengths, ...}*. *y* can have five classes {Weak (1), Not bad (2), Good (3), Great(4), Master Piece(5)}.

4. What is **Discretisation**, and where might it be used?

We have a (continuous) numeric attribute, but we wish to have a nominal (or ordinal) attribute. Some learners (like Decision Trees) generally work better with nominal attributes; some datasets inherently have groupings of values, where treating them as an equivalent might make it easier to discern underlying patterns.

5. Discretise the following dataset according to the (unsupervised) methods of **equal width** and **equal frequency**.

ID	A (°C)	B (mm)	C (hPa)	CLASS
1	22.5	4.6	1021.2	AUT
2	16.7	21.6	1027.0	AUT
3	29.6	0.0	1012.5	SUM
4	33.0	0.0	1010.4	SUM
5	13.2	16.4	1019.5	SPR
6	14.9	8.6	1016.4	SPR
7	18.3	7.8	995.4	WIN
8	16.0	5.6	1012.8	WIN

Equal width divides the range of possible values seen in the training set into equally– sized sub-divisions, regardless of the number of instances (sometimes 0!) in each division.

- For attribute C above, the largest value is 1027.0 and the smallest value is 995.4; the difference is 31.6:
- If we wanted two “buckets”, each bucket would be 15.8 wide, so that instances between 995.4 and 1011.2 take one value (4 and 7), and instances between 1011.2 and 1027.0 take another (1, 2, 3, 5, 6, and 8).
- If we wanted three “buckets”, each bucket would be about 10.5 wide; so that instances between 995.4 and 1005.9 take one value (just 7), instances between 1005.9 and 1016.4 take another value (3, 4, 6, and 8), and instances between 1016.4 and 1027.0 take yet another value (1, 2, and 5).

Equal frequency divides the range of possible values seen in the training set, such that (roughly) the same number of instances appear in each bucket.

- For attribute C above, if we sort the instances in ascending order, we find 7, 4, 3, 8, 6, 5, 1, 2.
- If we wanted two “buckets”, each bucket would have four instances; so that instances 7, 4, 3, and 8 would take one value, and the rest would take the other value.
- Sometimes, we also need to explicitly define the ranges, in case we obtain new instances later that we need to transform. There is some question about the intermediate values (between 1012.8 and 1016.4, in this case); typically, we place the dividing point at the median.