School of Computing and Information Systems
The University of Melbourne
COMP30027 Machine Learning (Semester 1, 2021)

Sample Solutions: Week 4

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

1. 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.

(i). Summarise some approaches to **supervised** discretisation.

Our general idea here is to sort the possible values, and create a nominal value for a region where most of the instances having the same label.

(ii). Discretise the above dataset according to the (unsupervised) methods of **equal width**, **equal frequency**, and **k-means** (breaking ties where necessary).

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.

k-means is actually a "clustering" approach, but it can work well in this context. If we want k buckets, we randomly choose *k* points to act as seeds. We then have an iterative approach where we: assign each instance to the bucket of the closest seed; update the "centroid" of the bucket with the mean of the values.

– For attribute C above, let's say we begin with two seeds: instance 3 (1012.5, bucket A) and instance 4 (1010.4, bucket B).

– Instance 1 (1021.2) is closer to A than B; 2 (1027.0) is closer to A; 3 is closer to A; 4 is closer to B; 5 (1019.5) is closer to A; 6 (1016.4) is closer to A; 7 (995.4) is closer to B; 8 (1012.8) is closer to A.

– We take the average of the values of instances 1, 2, 3, 5, 6, and 8 (1018.2) to be representative of cluster A, and the average of instances 4 and 7 (1002.9) to be representative of cluster B.

– Now, we iterate: 1 is still closer to A; 2 is still closer to A; 3 is still closer to A; 4 is still closer to B; 5 is still closer to A; 6 is still closer to A; 7 is still closer to B; 8 is still closer to A.

– Since this is the same assignment of values to clusters, we stop: instances 4 and 7 will have one value, and the other instances will have another value.

2. Find the (sample) **mean** and (sample) **standard deviation** for the attributes in the above dataset:

(i). In its entirety;

We have actually already been calculating sample means, in the centroid calculations for k-means. For example, the mean μ of the values of attribute C is determined by summing the values and dividing by the number of values:

$$\mu_c = \frac{1}{N}\sum c_i$$

$$= \frac{1}{8}(1021.2 + 1027.0 + 1012.5 + 1010.4 + 1019.5 + 1016.4 + 995.4 + 1012.8)$$

$$= \frac{1}{8}(8115.2) = 1014.4$$

The sample standard deviation is calculated by squaring the difference from the sample mean, dividing by 1 less than the number of values (this is a little surprising), and then taking the (positive) square root:

$$\sigma_c = \sqrt{\frac{\Sigma(c_i - \mu_c)^2}{(N-1)}} = \sqrt{\frac{46.24 + 158.76 + 3.61 + 16.0 + 26.01 + 4.0 + 361 + 2.56}{7}}$$

$$\approx \sqrt{88.31} \approx 9.40$$

(ii). For each individual class;

We would ideally do this with more instances!

(iii). How could we use this information when building a classifier over this data?

We could construct a Gaussian probability density function, which would allow us to estimate the probability of observing any given value, using pointwise estimation, or by counting the number of standard deviations it is from the mean

So for attribute C we can use the following normal distribution function:

$$f(x) = \frac{1}{\sigma_c\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{x-\mu_c}{\sigma_c})^2} = \frac{1}{9.4\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{x-1014.4}{9.4})^2}$$

3. How is **holdout** evaluation different to **cross-validation** evaluation? What are some reasons we would prefer one strategy over the other?

   In a holdout evaluation strategy, we partition the data into a training set and a test set: we build the model on the former, and evaluate on the latter.

   In a cross-validation evaluation strategy, we do the same as above, but a number of times, where each iteration uses one partition of the data as a test set and the rest as a training set (and the partition is different each time).

   Why we prefer cross-validation to holdout evaluation strategy? Because, holdout is subject to some random variation, depending on which instances are assigned to the training data, and which are assigned to the test data. Any instance that forms part of the model is excluded from testing, and vice versa. This could mean that our estimate of the performance of the model is way off, or changes a lot from data set to data set.

   While Cross–validation mostly solves this problem: we're averaging over a bunch of values, so that one weird partition of the data won't throw our estimate of performance completely off; also, each instance is used for testing, but also appears in the training data for the models built on the other partitions. It usually takes much longer to cross-validate, however, because we need to train a model for every test partition.

4. A **confusion matrix** is a summary of the performance of a (supervised) classifier over a set of development ("test") data, by counting the various instances:

		Actual			
		a	*b*	*c*	*d*
Classified	*a*	10	2	3	1
	b	2	5	3	1
	c	1	3	7	1
	d	3	0	3	5

(i). Calculate the classification **accuracy** of the system. Find the **error rate** for the system.

In this context, Accuracy is defined as the fraction of correctly identified instances, out of all of the instances. In the case of a confusion matrix, the correct instances are the ones enumerated along the main diagonal (classified as a and actually a etc.):

$$\begin{aligned} Accuracy &= \frac{\text{\# of correctly identified instance}}{total\ \#\ of\ instances} \\ &= \frac{10+5+7+5}{10+2+3+1+2+5+3+1+1+3+7+1+3+0+3+5} \\ &= \frac{27}{50} = 54\% \end{aligned}$$

Error rate is just the complement of accuracy:

$$Error\ Rate = \frac{\text{\# of incorrectly identified instance}}{total\ \#\ of\ instances} = 1 - Accuracy = 1 - 54\% = 46\%$$

(ii). Calculate the **precision**, **recall**, **F-score** (where $\beta = 1$), **sensitivity**, and **specificity** for class d. (Why can't we do this for the whole system? How can we consider the whole system?)

Precision for a given class is defined as the fraction of correctly identified instances of that class, from the times that class was attempted to be classified. We are interested in the true positives (TP) where we attempted to classify an item as an instance of said class (in this case, d) and it was actually of that class (d): in this case, there are 5 such instances. The false positives (FP) are those items that we attempted to classify as being of class d, but they were actually of some other class: there are 3 + 0 + 3 = 6 of those.

$$Precision = \frac{TP}{TP + FP} = \frac{5}{5 + 3 + 0 + 3} = \frac{5}{11} \approx 45\%$$

Recall for a given class is defined as the fraction of correctly identified instance of that class, from the times that class actually occurred. This time, we are interested in the true positives, and the false negatives (FN): those items that were actually of class d, but we classified as being of some other class; there are 1 + 1 + 1 = 3 of those.

$$Recall = \frac{TP}{TP + FN} = \frac{5}{5 + 1 + 1 + 1} = \frac{5}{8} \approx 62\%$$

F-score is a measure which attempts to combine Precision (P) and Recall (R) into a single score. In general, it is calculated as:

$$F_\beta = \frac{(1 + \beta^2)\ P.R}{(\beta^2.P) + R}$$

By far, the most typical formulation is where the parameter β is set to 1: this means that Precision and Recall are equally important to the score, and that the score is a harmonic mean.

In this case, we have calculated the Precision of class d to be 0.45 and the Recall to be 0.62. The F-score where ($\beta = 1$) of class d is then:

$$F_1 = \frac{2\ P.R}{P + R} = \frac{2 \times 0.45 \times 0.62}{0.45 + 0.62} \approx 53\%$$

Sensitivity is defined the same way as Recall: TP/(TP+FN)

Specificity is Precision with respect to the negative instances:

$$Specificity = \frac{TN}{TN + FP}$$

$$= \frac{10 + 2 + 3 + 2 + 5 + 3 + 1 + 3 + 7}{10 + 2 + 3 + 2 + 5 + 3 + 1 + 3 + 7 + 3 + 0 + 3} = \frac{36}{42} \approx 86\%$$