School of Computing and Information Systems
The University of Melbourne
COMP90049 INTROCTION to MACHINE LEARNING
(Semester 2, 2020)

Sample Solutions: Week 4

1. 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:

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

(ii). Calculate the **precision**, **recall** and **F-score** (where $\beta = 1$), 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:

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

2. For the following dataset:

ID	*Outl*	*Temp*	*Humi*	*Wind*	PLAY
		TRAINING INSTANCES			
A	s	h	n	F	N
B	s	h	h	T	N
C	o	h	h	F	Y
D	r	m	h	F	Y
E	r	c	n	F	Y
F	r	c	n	T	N
		TEST INSTANCES			
G	o	m	n	T	?
H	?	h	?	F	?

(i). Classify the test instances using the method of **0-R**.

0-R is the quintessentially baseline classifier: we throw away all of the attributes, other than the class labels, and just predict each test instance according to whichever label is most common in the training data. (Hence, it is also common called the "majority class classifier".)

In this case, the two labels Y and N are equally common in the training data — so we are required to apply a tie–breaker. Remember that we're simply choosing one label to be representative of the entire collection, and both labels seem equally good for that here: so, let's say N.

Consequently, both test instances are classified as N.

(ii). *[OPTIONAL]* Classify the test instances using the method of **1-R**.

1-R is a slightly better baseline, which requires us to choose a single attribute to represent the entire decision–making process. For example, if `Outl` is our preferred attribute, then we base the classification of each test instance solely based on its value of `Outl`. (This is sometimes called a "Decision Stump".)

Given our preferred attribute, we will label a test instance according to whichever label in the training data was most common, for training instances with the corresponding attribute value.

How do we decide which attribute to choose? Well, the most common method is simply by counting the errors made on the training instances.

Let's say we chose `Outl`: first, we need to observe the predictions for the 3 values (s, o, and r), and then we will count up the errors that those predictions will make on the training data (it turns out that we can do this simultaneously):

- When `Outl` is `s`, there are two such instances in the training data. Both of these in- stances are labelled as `N`: our label for this attribute value will be `N`. We will therefore predict the label of both of these training instances correctly (we will predict `N`, and both of these instances are actually `N`).
- When `Outl` is `o`, there is just a single instance in the training data, labelled as `Y`: our label for this attribute value will be `Y`. We will therefore predict the label of this training instance correctly.
- When `Outl` is `r`, there are three such instances in the training data. Two of these instances are labelled as `Y`, the other is `N`: our label for this attribute value will be `Y` (as there are more `Y` instances than `N` instances — you can see that we're applying the method of 0-R here). We will therefore make one error: the instance which was actually n.

In total, that is 1 error for `Outl`. Hopefully, you can see that this is a very wordy explanation of a very simple idea. (We will go through the other attributes more quickly.)

For `Temp`:

- When `Temp` is `h`, there are two `N` instances and one `Y`: we will make one mistake.
- When `Temp` is `m`, there is one `Y` instance: we won't make any mistakes.
- When `Temp` is `c`, there is one `N` instance and one `Y` instance: we will make one mistake.

This is 2 errors in total for `Temp`, which means that it's less good that `Outl`.

We'll leave the other attributes as an exercise, and assume that `Outl` was the best attribute (it's actually tied with `Wind`): how do the test instances get classified?

- For test instance `G`, `Outl` is `o` — the 0-R classifier over the `o` instances from the training data gives `Y`, so we predict `Y` for this instance.
- For test instance `H`, we don't have a value for `Outl` so we cannot find the label. But if we use the assumption of `Outl = s` (as suggested by the question) — the 0-R classifier over the `s` instances from the training data gives `N`, so we predict `N` for instance `H`.

3. Given the above dataset, build a Naïve Bayes model for the given training instances.

A Naïve Bayes model is probabilistic classification Model. All we need for building a Naive Bayes model is to calculate the right probabilities (Prior and Conditional).

For this dataset, our class (or label or variable we trying to predict) is `PLAY`. So we need the probability of each label (the prior probabilities):

$$P(\text{Play} = \text{Y}) = \frac{1}{2} \qquad P(\text{Play} = \text{N}) = \frac{1}{2}$$

We also need to identify all the conditional probabilities between the labels of class (`PLAY`) and all the other attribute values such as `s`, `o`, `r` (for `Outlook`) or `h`, `m`, `c` (for `Temp`) and so on:

P(Outl = s | N) = $\frac{2}{3}$	P(Outl = o | N) = 0	P(Outl = r | N) = $\frac{1}{3}$
P(Outl = s | Y) = 0	P(Outl = o | Y) = $\frac{1}{3}$	P(Outl = r | Y) = $\frac{2}{3}$
P(Temp = h | N) = $\frac{2}{3}$	P(Temp = m | N) = 0	P(Temp = c | N) = $\frac{1}{3}$
P(Temp = h | Y) = $\frac{1}{3}$	P(Temp = m | Y) = $\frac{1}{3}$	P(Temp = c | Y) = $\frac{1}{3}$
P(Humi = n | N) = $\frac{2}{3}$	P(Humi = h | N) = $\frac{1}{3}$	
P(Humi = n | Y) = $\frac{1}{3}$	P(Humi = h | Y) = $\frac{2}{3}$	
P(Wind = T | N) = $\frac{2}{3}$	P(Wind = F | N) = $\frac{1}{3}$	
P(Wind = T | Y) = 0	P(Wind = F | Y) = 1	

4. Using the Naïve Bayes model that you developed in question 4, classify the given test instances.

(i). No smoothing.

For instance **G** we have the following:

N: $$P(N) \times\ P(Outl = o \mid N)\ P(Temp = m \mid N) P(Humi = n \mid N)\ P(Wind = T \mid N)$$

$$= \frac{1}{2}\ \times\ 0\ \times 0\ \times\ \frac{2}{3} \times \frac{2}{3} = 0$$

Y: $$P(Y) \times\ P(Outl = o \mid Y)\ P(Temp = m \mid Y) P(Humi = n \mid Y)\ P(Wind = T \mid Y)$$

$$= \frac{1}{2}\ \times\ \frac{1}{3} \times \frac{1}{3} \times\ \frac{1}{3} \times 0 = 0$$

To find the label we need to compare the results for the two tested labels (`Y` and `N`) and find the one that has a higher likelihood (Maximum Likelihood Estimation).

$$\hat{y} =\ argmax_{y \in \{Y,N\}}\ P(y|T = G)$$

However, based on these calculations we find that both values are 0! So our model is unable to predict any label for test instance G.

The fact is as long as there is a single 0 in our probabilities, none of the other probabilities in the product really matter.

For **H**, we first observe that the attribute values for `Outl` and `Humi` are missing (?). In Naive Bayes, this just means that we calculate the product without those attributes:

N: $$P(N) \times\ P(Outl =?\mid N)\ P(Temp = h \mid N) P(Humi = ?\mid N)\ P(Wind = F \mid N)$$

$$\approx P(N) \times\ P(Temp = h \mid N) P\ P(Wind = F \mid N)$$

$$= \frac{1}{2} \times \frac{2}{3} \times \frac{1}{3} = \frac{1}{9}$$

Y: $$P(Y) \times \ P(Outl = ? \mid Y)\ P(Temp = h \mid Y) P(Humi = ? \mid Y)\ P(Wind = F \mid Y)$$

$$\approx P(N) \times \ P(Temp = h \mid Y) P\ P(Wind = F \mid Y)$$

$$= \frac{1}{2} \times \frac{1}{3} \times 1 = \frac{1}{6}$$

Therefore, the result of our argmax function for the test instance **H** is $\mathbf{Y}$.

$$argmax_{y \in \{Y,N\}}\ P(y|T = H) = Y$$

(ii). Using the "epsilon" smoothing method.

For test instance G, using the 'epsilon' smoothing method, we can simply replace the 0 values with a small positive constant (like 10^{-6}), that we call ε. So we'll have:

N: $$= \frac{1}{2} \times \varepsilon \ \times \varepsilon \ \times \frac{2}{3} \times \frac{2}{3} = \frac{2\varepsilon^2}{9}$$

Y: $$= \frac{1}{2} \times \frac{1}{3} \times \frac{1}{3} \times \frac{1}{3} \times \varepsilon = \frac{\varepsilon}{54}$$

By smoothing, we can sensibly compare the values. Because of the convention of ε being very small (it should be (substantially) less than $\frac{1}{6}$ (*6 is the number of training instances*)), Y has the greater score (higher likelihood). So Y is the output of our **argmax** function and **G is classified as Y.**

A quick note on the 'epsilons':

This isn't a serious smoothing method, but does allow us to sensibly deal with two common cases:

- Where two classes have the same number of 0s in the product, we essentially ignore the 0s.
- Where one class has fewer 0s, that class is preferred.

For **H**, we don't have any zero probability so the calculations are similar to when we had no smoothing:

N: $$P(N) \times \ P(Temp = h \mid N)\ P(Wind = F \mid N)$$

$$\approx P(N) \times \ P(Temp = h \mid N) P\ P(Wind = F \mid N)$$

$$= \frac{1}{2} \times \frac{2}{3} \times \frac{1}{3} = \frac{1}{9} \cong 0.1$$

Y: $$P(Y) \times \ \ P(Temp = h \mid Y)\ P(Wind = F \mid Y)$$

$$\approx P(N) \times \ P(Temp = h \mid Y) P\ P(Wind = F \mid Y)$$

$$= \frac{1}{2} \times \frac{1}{3} \times \frac{3}{3} = \frac{1}{6} \cong 0.16$$

Therefore, the result of our argmax function for the test instance **H** is $\mathbf{Y}$.

$$argmax_{y \in \{Y,N\}}\ P(y|T = H) = Y$$

(iii). *[OPTIONAL]* Using "Laplace" smoothing ($\alpha = 1$)

This is similar, but rather than simply changing the probabilities that we have estimated to be equal to 0, we are going to modify the way in which we estimate a conditional probability:

$$P_i = \frac{x_i + \alpha}{N + \alpha d}$$

In this method we add α, which is 1 here, to all possible event (seen and unseen) for each attribute. So all unseen event (that currently have the probability of 0) will receive a count of 1 and the count for all seen events will be increased by 1 to ensure that the monocity is maintained.

For example, for the attribute `Outl` that have 3 different values (`s`, `o`, and `r`). Before, we estimated *P(`Outl` =`o`|`Y`)* = $\frac{1}{3}$ before; now, we add 1 to the numerator (add 1 to the count of `o`), and 3 to the denominator (1 (for `o`) + 1 (for `r`)+ 1 (for `s`)). So now *P(`Outl` = o| Y)* have the estimate of $\frac{1+1}{3+3} = \frac{2}{6}$.

In another example, *P(`Wind` = T| Y)* is not presented (unseen) in our training dataset (*P(`Wind` = T| Y)* = $\frac{0}{3}$). Using the Laplace smoothing (α =1), we add 1 to the count of `Wind` = `T` (given `Play` = `Y`) and 1 to the count of `Wind` = `F` (given `Play` = `Y`) and so now we have *P(`Wind` = T| Y)* = $\frac{0+1}{3+2} = \frac{1}{5}$.

Typically, we would apply this smoothing process when building the model, and then substitute in the Laplace-smoothed values when making the predictions. For brevity, though, I'll make the smoothing corrections in the prediction step.

For G, this will look like:

$$N: \quad P(N) \times P(Outl = o \mid N)\, P(Temp = m \mid N) P(Humi = n \mid N)\, P(Wind = T \mid N)$$

$$= \frac{1}{2} \times \frac{0+1}{3+3} \times \frac{0+1}{3+3} \times \frac{2+1}{3+2} \times \frac{2+1}{3+2}$$

$$= \frac{1}{2} \times \frac{1}{6} \times \frac{1}{6} \times \frac{3}{5} \times \frac{3}{5} = 0.005$$

$$Y: \quad P(Y) \times P(Outl = o \mid Y)\, P(Temp = m \mid Y) P(Humi = n \mid Y)\, P(Wind = T \mid Y)$$

$$= \frac{1}{2} \times \frac{1+1}{3+3} \times \frac{1+1}{3+3} \times \frac{1+1}{3+2} \times \frac{0+1}{3+2}$$

$$= \frac{1}{2} \times \frac{2}{6} \times \frac{2}{6} \times \frac{2}{5} \times \frac{1}{5} \cong 0.0044$$

Unlike with the epsilon procedure, N has the greater score — even though there are two attribute values that have never occurred with N. So here **G is classified as N.**

For H:

$$N: \quad = \frac{1}{2} \times \frac{2+1}{3+3} \times \frac{1+1}{3+2} = 0.1$$

$$Y: \quad = \frac{1}{2} \times \frac{1+1}{3+3} \times \frac{3+1}{3+2} \cong 0.13$$

Here, Y has a higher score — which is the same as with the other method, which doesn't do any smoothing here — but this time it is only slightly higher.

5. *[OPTIONAL]* 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.