School of Computing and Information Systems
The University of Melbourne
COMP90049 Knowledge Technologies (Semester 2, 2018)
Workshop sample solutions: Week 9

1. For the following dataset:

apple	*ibm*	*lemon*	*sun*	CLASS	label	length
TRAINING INSTANCES						
4	0	1	1	FRUIT	F_1	$\sqrt{18}$
5	0	5	2	FRUIT	F_2	$\sqrt{54}$
2	5	0	0	COMPUTER	C_1	$\sqrt{29}$
1	2	1	7	COMPUTER	C_2	$\sqrt{55}$
TEST INSTANCES						
2	0	3	1	?	T_1	$\sqrt{14}$
1	0	1	0	?	T_2	$\sqrt{2}$

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

- For this part, we are interested in the (Euclidean) distances between the instances — this is more sensitive to the length of the instance (vector) than the cosine similarity, which may or may not be appropriate, depending on the data set.
- Recall the Euclidean distance between two points A and B:

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

- For T_1, we find the Euclidean distances to the four training instances:

$$\begin{aligned}
\text{dist}(F_1, T_1) &= \sqrt{\sum_k (F_{1,k} - T_{1,k})^2} \\
&= \sqrt{(4-2)^2 + (0-0)^2 + (1-3)^2 + (1-1)^2} \\
&= \sqrt{8} \approx 2.828 \\
\text{dist}(F_2, T_1) &= \sqrt{(5-2)^2 + (0-0)^2 + (5-3)^2 + (2-1)^2} \\
&= \sqrt{14} \approx 3.742 \\
\text{dist}(C_1, T_1) &= \sqrt{(2-2)^2 + (5-0)^2 + (0-3)^2 + (0-1)^2} \\
&= \sqrt{35} \approx 5.916 \\
\text{dist}(C_2, T_1) &= \sqrt{(1-2)^2 + (2-0)^2 + (1-3)^2 + (7-1)^2} \\
&= \sqrt{45} \approx 6.708
\end{aligned}$$

- With this distance metric, close neighbours are ones with low scores. If we use the 1-nearest neighbour method, we observe that the closest instance is F_1. This is a FRUIT instance, so we choose FRUIT for this test instance.

- For the second test instance:

$$\begin{aligned}
\text{dist}(F_1, T_2) &= \sqrt{\sum_k (F_{1,k} - T_{2,k})^2} \\
&= \sqrt{(4-1)^2 + (0-0)^2 + (1-1)^2 + (1-0)^2} \\
&= \sqrt{10} \approx 3.162 \\
\text{dist}(F_2, T_2) &= \sqrt{(5-1)^2 + (0-0)^2 + (5-1)^2 + (2-0)^2} \\
&= \sqrt{36} = 6.000 \\
\text{dist}(C_1, T_2) &= \sqrt{(2-1)^2 + (5-0)^2 + (0-1)^2 + (0-0)^2} \\
&= \sqrt{27} \approx 5.196 \\
\text{dist}(C_2, T_2) &= \sqrt{(1-1)^2 + (2-0)^2 + (1-1)^2 + (7-0)^2} \\
&= \sqrt{53} \approx 7.280
\end{aligned}$$

- Once more, the best instance is F_1, so we choose FRUIT.

(b) Using the **cosine similarity** measure, classify the test instances using the 3-NN method. Extend this to the **weighted** 3-NN method.

- We're using the cosine measure of similarity, interpretting the instances as vectors in the feature space, and we'll find the angles between the vectors to find the nearest neighbours among the training instances to each of the test instances.
- Recall that the cosine measure between two vectors A and B is calculated as:

$$\cos(A, B) = \frac{A \cdot B}{\mid A \mid \cdot \mid B \mid}$$

- Let's start by pre-calculating the lengths of the vectors (they're shown in the table above). For example:

$$\begin{aligned}
\mid F_1 \mid &= \sqrt{4^2 + 0^2 + 1^2 + 1^2} \\
&= \sqrt{18} \approx 4.24
\end{aligned}$$

- To find the nearest neighbours for T_1, we'll calculate the cosine measure for each of the

four training instances:

$$\begin{aligned}
\cos(F_1, T_1) &= \frac{F_1 \cdot T_1}{\mid F_1 \mid \cdot \mid T_1 \mid} \\
&= \frac{\langle 4,0,1,1\rangle \cdot \langle 2,0,3,1\rangle}{\mid \langle 4,0,1,1\rangle \mid \cdot \mid \langle 2,0,3,1\rangle \mid} \\
&= \frac{4 \cdot 2 + 0 \cdot 0 + 1 \cdot 3 + 1 \cdot 1}{\sqrt{18} \cdot \sqrt{14}} \\
&= \frac{12}{\sqrt{18} \cdot \sqrt{14}} \approx 0.7559 \\
\cos(F_2, T_1) &= \frac{F_2 \cdot T_1}{\mid F_2 \mid \cdot \mid T_1 \mid} \\
&= \frac{\langle 5,0,5,2\rangle \cdot \langle 2,0,3,1\rangle}{\mid \langle 5,0,5,2\rangle \mid \cdot \mid \langle 2,0,3,1\rangle \mid} \\
&= \frac{27}{\sqrt{54} \cdot \sqrt{14}} \approx 0.9820 \\
\cos(C_1, T_1) &= \frac{C_1 \cdot T_1}{\mid C_1 \mid \cdot \mid T_1 \mid} \\
&= \frac{\langle 2,5,0,0\rangle \cdot \langle 2,0,3,1\rangle}{\mid \langle 2,5,0,0\rangle \mid \cdot \mid \langle 2,0,3,1\rangle \mid} \\
&= \frac{4}{\sqrt{29} \cdot \sqrt{14}} \approx 0.1985 \\
\cos(C_2, T_1) &= \frac{C_2 \cdot T_1}{\mid C_2 \mid \cdot \mid T_1 \mid} \\
&= \frac{\langle 1,2,1,7\rangle \cdot \langle 2,0,3,1\rangle}{\mid \langle 1,2,1,7\rangle \mid \cdot \mid \langle 2,0,3,1\rangle \mid} \\
&= \frac{12}{\sqrt{55} \cdot \sqrt{14}} \approx 0.4324
\end{aligned}$$

- At this point, we consider the four values that we calculated. We're looking for the 3-best nearest neighbours: for the cosine measure, these are the instance with the greatest values. For T_1, this is F_2 with a score of 0.9820, and F_1 and C_2.
- Two of the 3-best neighbours are of class FRUIT, whereas only one is of class COMPUTER: we apply a voting procedure, and here FRUIT (with 2) out-votes COMPUTER (with 1), so we classify T_1 as FRUIT.
- What if we were using "weighted" k-nearest neigbour? Well, each of the k best neighbours gets a weighted vote, according to the cosine similarity score we calculated above. We then accumulate these weighted votes, and the class with the greater weighting is the one that we choose.
- In this case, FRUIT has a total weight of 1.7379 (from F_2 and F_1) and COMPUTER only 0.4324 (from C_2), so we choose FRUIT.

- T_2 is similar:

$$\begin{aligned}
\cos(F_1, T_2) &= \frac{F_1 \cdot T_2}{\mid F_1 \mid \cdot \mid T_2 \mid} \\
&= \frac{\langle 4,0,1,1\rangle \cdot \langle 1,0,1,0\rangle}{\mid \langle 4,0,1,1\rangle \mid \cdot \mid \langle 1,0,1,0\rangle \mid} \\
&= \frac{4 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 0}{\sqrt{18} \cdot \sqrt{2}} \\
&= \frac{5}{\sqrt{18} \cdot \sqrt{2}} \approx 0.8333 \\
\cos(F_2, T_2) &= \frac{F_2 \cdot T_2}{\mid F_2 \mid \cdot \mid T_2 \mid} \\
&= \frac{\langle 5,0,5,2\rangle \cdot \langle 1,0,1,0\rangle}{\mid \langle 5,0,5,2\rangle \mid \cdot \mid \langle 1,0,1,0\rangle \mid} \\
&= \frac{10}{\sqrt{54} \cdot \sqrt{2}} \approx 0.9623 \\
\cos(C_1, T_2) &= \frac{C_1 \cdot T_2}{\mid C_1 \mid \cdot \mid T_2 \mid} \\
&= \frac{\langle 2,5,0,0\rangle \cdot \langle 1,0,1,0\rangle}{\mid \langle 2,5,0,0\rangle \mid \cdot \mid \langle 1,0,1,0\rangle \mid} \\
&= \frac{2}{\sqrt{29} \cdot \sqrt{2}} \approx 0.2626 \\
\cos(C_2, T_2) &= \frac{C_2 \cdot T_2}{\mid C_2 \mid \cdot \mid T_2 \mid} \\
&= \frac{\langle 1,2,1,7\rangle \cdot \langle 1,0,1,0\rangle}{\mid \langle 1,2,1,7\rangle \mid \cdot \mid \langle 1,0,1,0\rangle \mid} \\
&= \frac{2}{\sqrt{55} \cdot \sqrt{2}} \approx 0.1907
\end{aligned}$$

- Here again, F_2 is the nearest neighbour, followed by F_1 and C_2.
- Hence, in the unweighted version, we choose FRUIT ($2 > 1$), and in the weighted version, we still choose FRUIT ($1.7956 > 0.2626$).

2. For the following dataset:

apple	*ibm*	*lemon*	*sun*	CLASS
TRAINING INSTANCES				
Y	N	Y	Y	FRUIT
Y	N	Y	Y	FRUIT
Y	Y	N	N	COMPUTER
Y	Y	Y	Y	COMPUTER
TEST INSTANCES				
Y	N	Y	Y	?
Y	N	Y	N	?

Use the method of **Naive Bayes** classification, as shown in lectures, to classify the test instances. Revise some of the assumptions that are built into the model.

- The assumption most central to the formulation of the Naive Bayes classifier that we have discussed is the **conditional independence assumption**, namely, that each attribute is independent to all of the other attributes, given the class under consideration. This assumption (while false) is necessary to make the problem tractable, where finding reliable estimates of the joint distribution of features requires more data than we are likely to have.

- Another important assumption is in the way we find probabilities from the training data. This is most important for the **maximum likelihood estimation** we perform over the class priors (in contrast, we hedge our bets on the posterior probabilities of the terms using **smoothing**).
- The fact that these assumptions are demonstrably untrue makes it seem like the classifier should not be effective at predicting the classes of unseen data. However, Naive Bayes is a pretty solid performer! It turns out that the methodology is robust enough to produce decent predictions, despite small (and predictable) discrepancies in the individual probabilities under consideration.
- There are also a number of more minor assumptions:
  - We assume that the distribution of classes in the test set is (roughly) the same as the distribution of classes in the training set, and also that all of the classes in the test data are attested in the training data. This is often phrased as "the training and test instances were sampled from the same underlying distribution." (In fact, this is an assumption of most supervised machine learning algorithms.)
  - We typically require some kind of assumption for our smoothing method, depending on which smoothing method we actually use.
  - We need to make some assumptions about the distribution of continuous attributes (typically Gaussian) if they exist in our data set.
- Anyway, let's classify these test instances:
- Naive Bayes selects a class c from a set of classes C for a test instance $T = \langle t_1, t_2, \ldots, t_n \rangle$ using a set of training instances $\mathcal{D}$ according to:

$$c = \text{argmax}_{c_j \in C} P(c_j) P(T \mid c_j) \tag{1}$$

- We will expand $P(T \mid c_j)$ based on our conditional independence assumption (according the attribute values t_i seen in our test instance):

$$P(T \mid c_j) \approx \prod P(t_i \mid c_j) \tag{2}$$

- We can calculate the prior probabilities of the classes straight from the training data, using maximum likelihood estimation. For FRUIT, 2 of the 4 instances are FRUIT:

$$P(f) = \frac{2}{4} = \frac{1}{2}$$

- We find the conditional probabilities $P(t_i \mid c_j)$ based on maximum likelihood estimation from the data set, smoothing by replacing 0 values with a small, positive, non-zero value ϵ. (In the real world, we would probably use Laplacian smoothing, or some other more serious smoothing method.)
- To do this, we will observe what proportion of the instances for the given class contain the term that we're looking for. For example, for $P(a = \text{Y} \mid f)$: of the two FRUIT instances, both of them contain *apple*.

$$\begin{aligned} P(a = \text{Y} \mid f) &= \frac{2}{2} = 1 \\ P(i = \text{Y} \mid f) &= \frac{0}{2} = 0 \end{aligned}$$

$$
\begin{aligned}
P(l = \text{Y} \mid f) &= \frac{2}{2} = 1 \\
P(s = \text{Y} \mid f) &= \frac{2}{2} = 1 \\
P(a = \text{Y} \mid c) &= \frac{2}{2} = 1 \\
P(i = \text{Y} \mid c) &= \frac{2}{2} = 1 \\
P(l = \text{Y} \mid c) &= \frac{1}{2} \\
P(s = \text{Y} \mid c) &= \frac{1}{2}
\end{aligned}
$$

- When we substitute these values into (2) above, we will replace 0 values with ϵ.
- We will also need the conjugate probabilities, for example $P(t = \text{F} \mid c)$. To find these, we will observe that $P(a \mid b) + P(\bar{a} \mid b) = 1$. Consequently, $P(t = \text{N} \mid c) = 1 - P(t = \text{Y} \mid c)$.
- Going all the way back to (1), and substituting our simplification from (2), we will now consider the values for FRUIT and COMPUTER (they aren't really probabilities any more, but it isn't important now).

$$
\begin{aligned}
\text{FRUIT} \quad : \quad & P(a = \text{Y} \mid f)P(i = \text{N} \mid f)P(l = \text{Y} \mid f)P(s = \text{Y} \mid f)P(f) \\
= \quad & P(a = \text{Y} \mid f)(1 - P(i = \text{Y} \mid f))P(l = \text{Y} \mid f)P(s = \text{Y} \mid f)P(f) \\
= \quad & 1 \times (1 - 0) \times 1 \times 1 \times \frac{1}{2} \\
= \quad & \frac{1}{2} \\
\text{COMP} \quad : \quad & P(a = \text{Y} \mid c)P(i = \text{N} \mid c)P(l = \text{Y} \mid c)P(s = \text{Y} \mid c)P(c) \\
= \quad & P(a = \text{Y} \mid c)(1 - P(i = \text{Y} \mid c))P(l = \text{Y} \mid c)P(s = \text{Y} \mid c)P(c) \\
= \quad & 1 \times (1 - 1) \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \\
\approx \quad & 1 \times \epsilon \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \\
\approx \quad & \frac{\epsilon}{8}
\end{aligned}
$$

- For this test instance, T_1, we choose FRUIT, because $\frac{1}{2} > \frac{\epsilon}{8}$ (because ϵ is small).
- For T_2, the calculations are similar, except for the fact that $s = \text{N}$:

$$
\begin{aligned}
\text{FRUIT} \quad : \quad & P(a = \text{Y} \mid f)P(i = \text{N} \mid f)P(l = \text{Y} \mid f)P(s = \text{N} \mid f)P(f) \\
= \quad & P(a = \text{Y} \mid f)(1 - P(i = \text{Y} \mid f))P(l = \text{Y} \mid f)(1 - P(s = \text{Y} \mid f))P(f) \\
= \quad & 1 \times (1 - 0) \times 1 \times (1 - 1) \times \frac{1}{2} \\
\approx \quad & \frac{\epsilon}{2} \\
\text{COMP} \quad : \quad & P(a = \text{Y} \mid c)P(i = \text{N} \mid c)P(l = \text{Y} \mid c)P(s = \text{N} \mid c)P(c) \\
= \quad & P(a = \text{Y} \mid c)(1 - P(i = \text{Y} \mid c))P(l = \text{Y} \mid c)(1 - P(s = \text{Y} \mid c))P(c) \\
= \quad & 1 \times (1 - 1) \times \frac{1}{2} \times (1 - \frac{1}{2}) \times \frac{1}{2} \\
\approx \quad & \frac{\epsilon}{8}
\end{aligned}
$$

- And again, this is classified as FRUIT.

(a) [EXTENSION] Revise the **multinomial distribution**. Naive Bayes can be extended to account for integer frequencies in the data (like in Question 1) using this model. Read up on so-called **multinomial Naive Bayes**.

3. Revise Support Vector Machines, paying particular attention to the terms "linear separability" and "maximum margin".

   - Support vector machines attempt to partition the training data based on the best line (hyperplane) that divide the positive instances of the class that we're looking for from the negative instances.
   - We say that the data is *linearly separable*, when the points in our k-dimensional vector space corresponding to positive instances can indeed be separated from the negative instances by a line (or, more accurately, a $(k-1)$-dimensional hyperplane). This means that all of the positive instances are on one side of the line, and all of the negative instances are on the other side.
   - What is the definition of "best" line? Well, in fact, we choose a pair of parallel lines, one for the positive instances, and one for the negative instances. The pair that we choose is the one that has the greatest perpendicular distance between these parallel lines (calculated using the normal; because the lines are parallel, they have the same normal) — this is called the "margin".
   - These two lines are called the "support vectors" — we can classify test instances by calculating (again, using the normal) which of the support vectors is closer to the point defined by the test instance. Alternatively, we can just use a single line, halfway between the two support vectors, and use the normal to calculate which "side" of the line the test instance is on.

   (a) What are "soft margins", and when are they desirable?
      - Soft margins are when we relax the notion of linear separability. A small number of points are allowed to be on the "wrong" side of the line, if we get a (much) better set of support vectors that way (i.e. with a (much) larger margin).
      - Sometimes the data is *almost* linearly separable — if we accept that we probably aren't going to classify every single test instance correctly anyway, we can produce a classifier that hopefully generalises better to unseen data.

   (b) Why are we interested in "kernel functions" here?
      - The so-called "kernel trick" is often used to transform data.
      - For Support Vector Machines, sometimes the data isn't linearly separable, but after applying some kind of function — where some useful properties of the data remain (for example, monotonicity of the inner product) — the data becomes linearly separable. If we do this, we can apply the algorithm as usual.

   (c) Why are SVMs "binary classifiers", and how can we extend them to "multi-class classifiers"?
      - We're trying to find a line that partitions the data into true and false, suitably interpretted for our data.
      - The point is that for two classes, we only need a single line. (Or more, accurately, a pair of parallel support vectors.)
      - We need (at least) two lines to partition data into three classes. For example, $x < 0$, $0 < x < 1$, $x > 1$ might be a useful partition. But what happens when these lines aren't parallel?
      - In general, we might want to find three "half–lines", radiating out from some central point between the (points corresponding to) the three different classes. The problem? This is numerically much more difficult (and usually intractable) to solve.
      - For problems with more than three classes, the entire notion of separating the data becomes poorly-defined — unless we wish to fall back to some clustering approach, in which case, we might as well use Nearest Prototype.
      - In practice, then, we don't try to do the above, but rather build *multiple* (binary) SVMs, either by comparing the instances of one class against the instances of all of the other classes (one-vs-many) and then choosing the most confident prediction, or by considering each pair of classes (one-vs-one) and then choosing the class that comes up the most frequently.