School of Computing and Information Systems
The University of Melbourne
COMP90049 Knowledge Technologies (Semester 1, 2017)
Workshop exercises: Week 12

1. 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 divides 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 is the significance of allowing "some margin of errors", indicated by ξ in the lectures?

   - Also known as "soft margins", this is 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 probably 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 <u>much</u> 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.

- Instead, we are typically required to build multiple models, either by comparing each class against all of the other classes ("one versus many") or by building a model for each pair of classes ("one versus one").

2. What is **Clustering**?

- Clustering is the most fundamental form of **unsupervised** machine learning: in the absence of labelled (training) instances, we "group" instances together based on their similarity.
- To make a more direct parallel to supervised machine learning: a group of similar instances ("cluster") can be regarded as having the same "label".

(a) What is the difference between "partitional" and "hierarchical" clustering? What are some other distinctions that we can draw between clusterings?

- A partitional ("flat") clustering is when the instances are grouped so that each instance belongs to a single cluster; a hierarchical clustering is nested, so that an instance in one cluster also belongs to the surrounding clusters ("higher up" in the tree, or dendrogram).
- Clusterings can also be fuzzy (or probabilistic); they can be partial (not all of the data is clustered); they can be heterogeneous (the data isn't always clustered the same way); and so on.

(b) How does the k**-means** algorithm cluster data? Given the following dataset:

id	*apple*	*ibm*	*lemon*	*sun*
A	4	0	1	1
B	5	0	5	2
C	2	5	0	0
D	1	2	1	7
E	2	0	3	1
F	1	0	1	0

Apply k-means, using the Manhattan distance, and seeds A and D. What would happen if we had used different instances as seeds?

- In k-means, we have an iterative process where we assign instances according to cluster having the nearest centroid, and then re-calculating the centroids according to the instances in the cluster.
- We begin by setting the initial centroids for our two clusters, let's say cluster 1 has centroid $C_1 = \langle 4, 0, 1, 1\rangle$ and cluster 2 $C_2 = \langle 1, 2, 1, 7\rangle$.
- We now calculate the distance for each instance ("training" and "test" are equivalent in this context) to the centroids of each cluster:

$$\begin{aligned}
d(A, C_1) &= \mid 4-4 \mid + \mid 0-0 \mid + \mid 1-1 \mid + \mid 1-1 \mid \\
&= 0 \\
d(A, C_2) &= \mid 4-1 \mid + \mid 0-2 \mid + \mid 1-1 \mid + \mid 1-7 \mid \\
&= 11 \\
d(B, C_1) &= \mid 5-4 \mid + \mid 0-0 \mid + \mid 5-1 \mid + \mid 2-1 \mid \\
&= 6 \\
d(B, C_2) &= \mid 5-1 \mid + \mid 0-2 \mid + \mid 5-1 \mid + \mid 2-7 \mid \\
&= 15 \\
d(C, C_1) &= \mid 2-4 \mid + \mid 5-0 \mid + \mid 0-1 \mid + \mid 0-1 \mid \\
&= 9 \\
d(C, C_2) &= \mid 2-1 \mid + \mid 5-2 \mid + \mid 0-1 \mid + \mid 0-7 \mid \\
&= 12
\end{aligned}$$

$$\begin{aligned}
d(D, C_1) &= |\,1-4\,| + |\,2-0\,| + |\,1-1\,| + |\,7-1\,| \\
&= 11 \\
d(D, C_2) &= |\,1-1\,| + |\,2-2\,| + |\,1-1\,| + |\,7-7\,| \\
&= 0 \\
d(E, C_1) &= |\,2-4\,| + |\,0-0\,| + |\,3-1\,| + |\,1-1\,| \\
&= 4 \\
d(E, C_2) &= |\,2-1\,| + |\,0-2\,| + |\,3-1\,| + |\,1-7\,| \\
&= 11 \\
d(F, C_1) &= |\,1-4\,| + |\,0-0\,| + |\,1-1\,| + |\,0-1\,| \\
&= 4 \\
d(F, C_2) &= |\,1-1\,| + |\,0-2\,| + |\,1-1\,| + |\,0-7\,| \\
&= 9
\end{aligned}$$

- We now assign each instance to the cluster with the smallest (Manhattan) distance to the cluster's centroid: for `A`, this is C_1 because $0 < 11$, for `B`, this is C_1 because $6 < 15$, and so on. It turns out that `A`, `B`, `C`, `E`, and `F` all get assigned to cluster 1, and `D` is assigned to cluster 2.
- We now update the centroids of the clusters, by calculating the arithmetic mean of the attribute values for the instances in each cluster. For cluster 1, this is:

$$\begin{aligned}
C_1 &= \langle \frac{4+5+2+2+1}{5}, \frac{0+0+5+0+0}{5}, \frac{1+5+0+3+1}{5}, \frac{1+2+0+1+0}{5} \rangle \\
&= \langle 2.8, 1, 2, 0.8 \rangle
\end{aligned}$$

- For cluster 2, we're just taking the average of a single value, so obviously the centroid is just $\langle 1, 2, 1, 7 \rangle$.
- Now, we re-calcuate the distances of each instance to each centroid:

$$\begin{aligned}
d(A, C_1) &= |\,4-2.8\,| + |\,0-1\,| + |\,1-2\,| + |\,1-0.8\,| \\
&= 3.4 \\
d(B, C_1) &= |\,5-2.8\,| + |\,0-1\,| + |\,5-2\,| + |\,2-0.8\,| \\
&= 7.4 \\
d(C, C_1) &= |\,2-2.8\,| + |\,5-1\,| + |\,0-2\,| + |\,0-0.8\,| \\
&= 7.6 \\
d(D, C_1) &= |\,1-2.8\,| + |\,2-1\,| + |\,1-2\,| + |\,7-0.8\,| \\
&= 10 \\
d(E, C_1) &= |\,2-2.8\,| + |\,0-1\,| + |\,3-2\,| + |\,1-0.8\,| \\
&= 3 \\
d(F, C_1) &= |\,1-2.8\,| + |\,0-1\,| + |\,1-2\,| + |\,0-0.8\,| \\
&= 4.6
\end{aligned}$$

- (In this case, the distance of each instance to cluster 2 hasn't changed, because the value of the centroid is the same as the previous iteration.)
- Now, we re-assign instances to clusters, according to the smaller (Manhattan) distance: `A` gets assigned to cluster 1 (because $3.4 < 11$), `B` gets assigned to cluster 1 (because $7.4 < 15$), and so on. In all, `A`, `B`, `C`, `E`, and `F` get assigned to cluster 1, and `D` to cluster 2.
- At this point, we observe that the assignments of instances to clusters is the same as the previous iteration, so we stop. (The newly-calculated centriods are going to be the same, so the algorithm has reached equilibrium.)

- The final assignment of instances to clusters here is: cluster 1 {`A,B,C,E,F`} and cluster 2 {`D`}.

3. For the following set of instances:

a_1	a_2	a_3	c
hot	windy	dry	Yes
mild	windy	rainy	No
hot	windy	rainy	Yes
cool	still	dry	Yes
cool	still	rainy	No
hot	still	dry	No
mild	still	dry	Yes

(a) Calculate the **confidence** and **support** of the Association Rule {still, Yes}→{dry}.

- The confidence of an association rule is calculated as: the count of instances where we see the elements in the (set of the) antecedent (left-hand side) *and* the consequent (right-hand side), divided by the count of instances where we see the consequent.
- Taking σ as the counting operator, and suitably interpretting the comma as "and", we can represent this formally as:

$$\text{Conf}(A \to B) \quad = \quad \frac{\sigma(A, B)}{\sigma(A)}$$

- Support is defined similarly, except that the denominator is the count of all of the instances N (or $\sigma(*)$):

$$\text{Supp}(A \to B) \quad = \quad \frac{\sigma(A, B)}{N}$$

- In this dataset, there are 7 instances, of which 4 have the attribute value {dry}, and 2 have all of {still, Yes, dry} (the fourth and seventh instancs). The confidence of the given association rule is therefore $\frac{2}{2} = 1$ and the support is $\frac{2}{7}$.

(b) Discuss how you would mine for effective **Association Rules**, according to some thresholds τ_c for confidence and τ_s for support.

- We want association rules with good confidence and good support, so we will attempt to extract every rule whose confidence and support are above both of these thresholds.
- One way to attempt this is to just generate every rule and calculate both support and confidence, but in general, there will be far too many rules (as the number of rules is largely exponential in the number of attributes).
- Instead, we will use the *a priori* algorithm. We need some terminology:
  - An *itemset* is a list of attribute values
  - The itemset corresponding to an association rule is the (set) union of the (set of) rules in the antecedent and the (set of) rules in the consequent. The reason that this is significant is that, when we calculate support, we don't care which side of the arrow a given attribute value is on.
  - An *n-itemset* is a list of n distinct attribute values
  - The association rules associated with an *n-itemset* are the (roughly) 2^n different rules that we can generate by variously placing some of the attribute values in the antecedent (set), and the rest in the consequent (set)
- Now, to restrict the number of rules under consideration, we observe that, from a given k-itemset, we can construct a number of $k+1$ itemsets by adding each one of the attribute values that aren't already in this rule — but that the support can only go **down** by adding another attribute value. (Why?)

- So, we start by considering all of the 1-itemsets (i.e. each attribute value in the collection): if any of them fail to pass our support threshold, then we can exclude that itemset from being a subset of an interesting 2-itemset.
- For example, if the support threshold is $\tau_s = 0.5$, the only interesting 1-itemsets are $\{$*still*$\}$, $\{$*dry*$\}$, and $\{$Yes$\}$. The only potentially interesting 2-itemsets are those that contain only those attribute values, namely $\{$*still,dry*$\}$, $\{$*still,*Yes$\}$, and $\{$*dry,*Yes$\}$.
- We would then calculate the supports of those itemsets (and continue to $\{$*still,dry,*Yes$\}$ if they continued to meet the support threshold of 0.5).
- Once we can no longer construct further itemsets that can meet the support threshold (because enough k itemsets have supports below the threshold so as to block all of the $k+1$ itemsets), we can generate all of the association rules from each itemset, calculate the confidences of these rules, and return with the ones that pass the confidence threshold τ_c.
- (We can construct about 2^k different association rules for a k-itemset. If we have many itemsets that meet the support threshold, there could still be too many rules to examine the confidences. In this case, we derive a similar procedure by observing that moving an attribute value from the antecedent to the consequent can only reduce the confidence of the rule. For a given k-itemset, we start with each of the k attribute values in the consequent one at a time, find the confidence, and proceed with rules that meet the confidence threshold.)