School of Computing and Information Systems
The University of Melbourne
COMP90049
Knowledge Technologies (Semester 1, 2018)
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.)
      - For three or more classes, a single line can't partition the data — we could try to find multiple lines, but there is no guarantee that they would be parallel. This problem is numerically much more difficult (and usually intractable) to solve. We could try to fall back to a clustering approach, but in that case, we might as well be using a far simpler supervised learner, like 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:

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

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

- 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`}.