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

Solution: Week 6

1) Revise SVMs, particularly the notion of "linear separability".

(i). If a dataset isn't linearly separable, an SVM learner has **two** major options. What are they, and why might we prefer one to the other?

1. Soft margins:

we permit a few points to be on the "wrong" side of the line, at a penalty, if this allows us to find a better (wider) margin.

If we suspect that the data is essentially linearly separable, except that a few points will be mis-classified (for example, if there are outliers in the data), then using a soft-margin SVM is the better choice. The kernel trick is too powerful to account for just a few instances (and, in the worst case, will generate a spurious transformation of the data to deal with them).

2. Kernel methods:

We (effectively) transform the data into a higher-dimensional space. Conceptually, this is by creating new dimensions (and consequently, new attributes), but the "kernel trick" allows us to do this without making the transformation explicit (and so, saving us some computation time).

If we suspect that the data isn't really linearly separable — because the instance topology isn't linear, but polynomial or circular or something — then the soft margin SVM will produce a spurious margin, because there really isn't a way to draw a straight line through the data at all. If we do, the model is too simplistic and results in underfitting.

(ii). 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 interpreted 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.

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.

(iii). Contrary to many geometric methods, SVMs work better (albeit slower) with large attribute sets. Why might this be true?

The clearest way of seeing this is to imagine that the dataset has a number of useful attributes (as in, they are strongly or moderately predictive of the class), and a number of not-so-useful attributes (as in, they are marginally predictive, or not predictive at all).

Most geometric methods calculate some kind of similarity or distance metric, which assumes that every attribute is equally important. For example, using the Manhattan Distance, the difference between the values of the useful attributes can be small, but if the difference between the not-so-useful attributes is large, then the two instances will be not-so-similar (they will be far apart in space)[1].

A typical SVM gets around this problem by (effectively) finding different "weights" for each attribute[2] — these form the normal vector *w*; the weights of the not-so-useful attributes will (hopefully) be negligible, compared to the weights of the useful attributes, so that the basis for the prediction (hopefully) becomes more meaningful.

There are other issues too, like how the SVM is building a linear combination (or non- linear combination, if we are using a non-linear kernel) of the attributes, rather than treating them all as completely distinct.

2) We have now seen a decent selection of (supervised) learners:

- Naive Bayes
- 0-R
- 1-R
- Decision Trees
- *k*-Nearest Neighbour
- Support Vector Machines

(i). For each, identify the model built during training.

– For NB, this is a set of prior probabilities $P(c_j)$ and a set of conditional probabilities $p(a_k|c_j)$ for each feature-value-class combination

– For 0-R, this is a class distribution, or simply the label of the most frequent class

– For 1-R, this is an attribute, and the majority class of the instances (a label) for each value that this attribute can take

– For a Decision Tree, this is the tree itself: every non-terminal node is labelled with an attribute, and each branch with an attribute value; every leaf is labelled with a class

– For k-NN, this is just the dataset itself!

– For an SVM, this is the maximum-margin hyperplane, defined by w and b, or equivalently by a set of Lagrange multipliers α_k, and the corresponding training instances for which $\alpha_k > 0$.

(ii). Rank the learners (approximately) by how fast they can classify a large set of test instances. (Note that this is largely independent of how fast they can build a model, and how well they work in general!)

[1] In fact, as the attribute set becomes large, there is a statistical bias to the distance measures producing values that make instances indistinguishable! In ML, this is known as the **curse of dimensionality**.

[2] There are improvements to *k*-NN which attempt to follow a similar logic, creating a vector space that isn't orthonormal.

Given N training instances, C classes, and D attributes, the amount of time required to make a prediction for each test instance would be as follows:

- The fastest few are clear: *0-R (O(0))*, *1-R (O(1))*, and Decision Trees *(O(D))*.
- The next is: Naïve Bayes *(O(CD + C))*.
- An SVM isn't built to deal with multiple classes directly.
  - If there are exactly two classes, it is *O(D+1)* (very similar to Decision Tree *(O(D))*.
  - If there are many classes, and we are using a one-vs-all strategy for building our SVM(s), it is approximately *O(CD + C)* the same as Naïve Bayes. If we are using one-vs-one, it is somewhat slower *(O($C^2D + C^2$)).*
- k-NN is much, much slower, because it must make a comparison for each instance *(O(ND + k)).*