School of Computing and Information Systems
The University of Melbourne
COMP30027 MACHINE LEARNING (Semester 1, 2019)

Tutorial exercises: Week 5

1. For the following dataset:

apple	*ibm*	*lemon*	*sun*	CLASS
TRAINING INSTANCES				
4	0	1	1	FRUIT
5	0	5	2	FRUIT
2	5	0	0	COMPUTER
1	2	1	7	COMPUTER
TEST INSTANCES				
2	0	3	1	?
1	2	1	0	?

   (a) Classify the test instances according to the method of **Nearest Prototype**.

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

   (c) Using the **Manhattan distance** measure, classify the test instances using the 3-NN method, for the three weightings we discussed in the lectures: majority class, inverse distance, inverse linear distance.

   (d) Can we do weighted k-NN using **cosine similarity**?

2. Revise SVMs, particularly the notion of "linear separability".

   (a) 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?

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

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

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

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

   (b) 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. For the following dataset:

	apple	*ibm*	*lemon*	*sun*	CLASS
			TRAINING INSTANCES		
A	4	0	1	1	FRUIT
B	5	0	5	2	FRUIT
C	2	5	0	0	COMPUTER
D	1	2	1	7	COMPUTER
			TEST INSTANCES		
	2	0	3	1	?
	1	2	1	0	?

(a) Classify the test instances according to the method of **Nearest Prototype**.

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

(c) Using the **Manhattan distance** measure, classify the test instances using the 3-NN method, for the three weightings we discussed in the lectures: majority class, inverse distance, inverse linear distance.

(d) Can we do weighted k-NN using **cosine similarity**?

1. (a) Nearest Prototype:

Step 1. Construct prototype for each class:

$$P_{class_i} = < \frac{\Sigma(A_1 | Class_i)}{\# \ Class_i}, \ldots, >$$

Step 2. use Distance Formula to Classify.

$$P_f = < \frac{4+5}{2}, \frac{0+0}{2}, \frac{1+5}{2}, \frac{1+2}{2} > = < 4.5, 0, 3, 1.5 >$$

$$P_c = < \frac{2+1}{2}, \frac{5+2}{2}, \frac{0+1}{2}, \frac{0+7}{2} > = < 1.5, 3.5, 0.5, 3.5 >$$

Use "Euclidean Distance" (Manhattan is similar)

$$d_E = \sqrt{\sum_K (a_k - b_k)^2}$$

For the first test instance.

$$d_E(T_1, P_f) = \sqrt{(2-4.5)^2 + (0-0)^2 + (3-3)^2 + (1-1.5)^2} = \sqrt{6.5} \checkmark$$

$$d_E(T_1, P_c) = \sqrt{(2-1.5)^2 + (0-3.5)^2 + (3-0.5)^2 + (1-3.5)^2} = \sqrt{25}$$

(b) Euclidean Distance + 1-NN

$D_E(T_1, A) = \sqrt{8} \rightarrow$ fruit

$D_E(T_1, B) = \sqrt{14}$

$D_E(T_1, C) = \sqrt{35}$

$D_E(T_1, D) = \sqrt{45}$

(c) Mahattan Distance + 3-NN

· $d_M(A, B) = \sum_k |a_k - b_k|$

For the first test instance.

top-3 smallest:

$D_M(T_1, A) = 4$ A - fruit

$D_M(T_1, B) = 6$ B - fruit

$D_M(T_1, C) = 9.$ C - computer

i) Mojority Class.

test 1 : fruit

ii) Inverse Distance. (ID)

1. $ID = \frac{1}{D+\varepsilon} \leftarrow \varepsilon = 1$

A : $\frac{1}{4+1} = \frac{1}{5}$ B : $\frac{1}{6+1} = \frac{1}{7}$ C: $\frac{1}{1+9} = \frac{1}{10}$

2. get class total score

fruit : A + B = $\frac{1}{5} + \frac{1}{7}$ = 0.34 Computer: C = 0.1

test 1: fruit

iii) Inverse linear Distance (ILD)

1. $W_j = \frac{d_k - d_j}{d_k - d_1}$ d_k: furthest neigh d_1: nearest neigh

A: $\frac{9-4}{9-4} = 1$ C: $\frac{9-9}{9-4} = 0$

B: $\frac{9-6}{9-4} = \frac{3}{5}$

2. get total score

fruit: $1 + \frac{3}{5} = \frac{8}{5}$

computer: 0

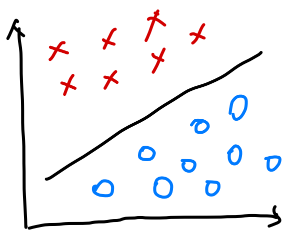
test 1: fruit

(d) Yes. this is easier than Calculate distance. because we assign a weighting for each instance using the cosine similarity directly. An overall weighting for a class can be obtained by summing the cosine scores for the instances of the corresponding class, from among the set of nearest neighbors.

2. Revise SVMs, particularly the notion of "linear separability".

   (a) 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?

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



(a) Option 1: Soft margins: We permit a few points in "wrong" side to find a better margin.

Option 2: Kernel methods: transform data into higher dimensional space.

前提 1: suspect is linear separable.

· a few points mis-classified → soft margins

· a few instances → kernel

前提 2: Suspect not linear separable.

soft margin → very wrong margin.

(b) the dataset has a number of useful and useless attributes.

Most geometric methods calculate Similarity | distance which assume all attributes are same important, but some non-relevant attributes will affect result a lot.

In SVMs, finding "different" weights for each attribute.

high on important, low on useless.

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

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

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

(b) 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!)

(a) Naive Bayes : a set of prior prob (P_{c_j}) and a set of posterior prob ($P(a_k|c_j)$)

0-R : looks class Distribution

1-R : this is an attribute, and the majority class

DT : a Tree, every none-terminal node is labelled with an attribute.

KNN: the data set itself.

NP: this is prototype for class.

SVM : hyperplane.

(b) N - training instances. C - classes D - attributes

time to make prediction:

fastest : 0-R: $O(0)$ 1-R $O(1)$ DT: $O(D)$

NP : $O(CD)$ NB : $O(CD+C)$

SVM One vs. one

$O(C^2D+C^2)$

slowest k-NN : $O(ND+k)$