section A

1. True
2. 1 \* 0.2 + 2 \* 0.3 + 3 \* 0.1 + 4 \* -0.7 = -1.7 < 1.5, thus it will give class 0
3. class 0, because choosing high k (more than the class 1 k) will result into a bias toward class 0
4. Split data into n chunks, train on n-1 chunk and test on the one that is left out, rotate and test again to see which parameter affect our training data and use that. The aim is to make sure all data has equal chance of appearing in the training or testing set
5. SVM nonono
6. Supervised learning: examples are labeled  
   Unsupervised learning: examples aren’t labeled
7. Naiive assume all condition are independent of each other
8. Finding similarity between data, and grouping similar data in clusters
9. D = (1,1,1,1)
10. 2

---

Section B

// this question appeared every single year, you better memorise it

function TreeBuilder(Subform, depth)

// base case

if(depth == 0) or (all examples in the subform have same label)

return the most common label

// recursive case

for each feature in the subform

try splitting the data

calculate information gain (entropy)

end for

choose the feature that gives most information gain // lowest entropy after (entropy before the split - entropy after the split))

split subform into 2 parts left and right

leftTree ← TreeBuilder(leftSubform, depth -1)

rightTree ← TreeBuilder(rightSubform, depth -1)

return tree

end function

the most information gain is done by calculating the entropy at every split point and after split and taking the feature that minimise the entropy.

Entropy is calculated as:

----

b)  
Entropy is calculated as:

x = {0, 1, 1, 1, 1, 0}

H(x) = - (p(x0)log(px0) + p(x1)log(px1))

= - (2/6 \* log(2/6) + 4/6 log (4/6))   
 = 0.276 based on log 10 (because my calculator doesn’t have log 2 T\_T)

---

c) // important

Ensemble: fits multiple model to training data, called based learner, when clarification is needed those models are queried for their vote and majority vote will be taken.

2 examples; Bagging and Boosting

Bagging:

function Bagging(input trainingData + label, data count M)

for j = 1 until M

take bootstrap sample T’ from T

build a model of T’

add model to set

end for

return set, when testing point x the majority vote is used to classify the data

Boosting:

1- take bootstrap sample from dataset

2- train a model on this boostrap

3- see which example were wrong

4- upweight the hard examples, downweight easy one

5- repeat from step 2 until desired committee vote is reached

Bagging vs Boosting:

Bagging is parallel

Boosting is serial (sequential)

N.B Bootstrap:

Generating multiple dataset from an original dataset. Select n training example from total N with replacement.

-----------------------------

2)  
a) 126 + 23 + 32 + 69 = 250

b) 69 + 23 = 92

c)

error rate = 1 - (69 + 126 / 126 + 23 + 32 + 69) = 1 - 0.78 = 0.22

sensitivity = TP / TP + FN = 69 / 69+23 = 0.75

specificity = TN / TN + FP = 126 / 126 + 32 = 0.79

d)

cost = FN \* cost(FN) + FP \* cost (FP)

= 23 \* 1000 + 32 \* 0

= 23000

e)

i) Perceptron works like a neuron in brain, it takes input data d, and w weights, and a threshold t, it then take the sum of all weighted features in the data and compare it to the threshold, if it is equal or exceed it, the neuron fires, otherwise it doesn’t.

ii) the learning algorithm:

initialise weights with random numbers between -1 and 1

for n = 1 to number of iteration

for each training example (x, y)

calculate activation

for each weight

update weight by learning rule

end for

end for

end for

iii) it can solve linearly separable problems within finite numbers of iteration, non-linear will require a multi layer perceptron

iv) the learning rate in the learning rule

learning rule:

new weight = old weight + learning rate \* (trueLabel - output) \* input

learning rate is user set parameter to determine how fast the separating line converges, a small learning rate takes longer to finish but will find the correct value more accurately, while larger learning rate will do a big jumps which could result in skipping the optimal sweet spot.

number of iterations ensure that we find best learning rate, smaller learning rate might need more iterations

----------

Section C: 3- SVM aren’t covered anymore

4-

1. Agglomerative: works by starting with each object being its own cluster and then merge the nearest cluster creating a new cluster, updating the distance, and repeat until number of cluster reached or no more clusters to merge.   
     
   1- convert each object point into datamatrix  
   2- set each object as its own cluster   
   3- merge the two nearest clusters together  
   4- updata data matrix to reflect the newly created clster  
   5- repeat from step 3 until number of clusters is 1 or K is reached  
     
   Advantage: Doesn’t need to know the numbers of cluster in advance, knowing them can set the condition of exiting.   
   Disadvantage: computationally expensive, very sensitive to noise and outliers
2. Talk about features you want to use (search terms, tags, date, clicks, source, https, etc..)  
   What value of K you want to set (after running algorithm analyse it and choose correct K)   
   Drawback of your algorithm (slow, affected by outliners)  
   Advantages (doesn’t need to know number of K in advance)  
   How to test it (internal index and external index)  
   Distance matrix you want to use (minkowskai, with i-index determining the value of p)  
   Single, vs complete vs average link. (go for average because features are equally important)
3. k-nearest:  
     
   algorithm:   
   Testing point x   
   For each training point x’   
    compute distance(x, x’)  
   end for  
   sort distances  
   choose K smallest distances  
   assign most common class   
     
     
   K-means:   
     
   algorithm:   
   Start initially with K cluster seed points assigned to randomly to the data point  
   1- assign each data object to the nearest cluster seed point using the appropriate data measurement algorithm   
   2- recompute the seed point by averaging the points in each cluster creating centroids   
   3- go back to step 1 until no new assignments are made  
     
   K-means:  
   + runtime of O(tKn) where t number of iteration and K number of clusters, so it is fairly fast  
   - Need to know K in advance → solved by using some i-indexing to discover best possible value for K  
   - Sensitive to initial seed points → solved by generating a different seed point at each iteration and choose best result  
   - Unable to handle noise and outliers (use k-medoids instead)  
   - Only works on defined data (for categorical data use k-mode)  
   - use (CLARA for better performance)   
     
     
   K-Nearest:  
   + high accuracy  
   - Need to find optimal K through trial and error   
   - very computationally expensive (slow) → Solved: by not using it  
     
     
   K-nearest vs k-means:  
   K-nearest supervised learning, require data and labels   
   k-means: unsupervised learning, doesn’t require labels