DATA MINING

TEST 2

INSTRUCTIONS:

- this test consists of 4 questions
- you may attempt all 4 questions.
- maximum marks = 40

Question 1 (10 marks)

The entropy of a sample D with respect to a target variable of k possible classes is defined as:

$$H(D) = -\sum_{i=1}^{k} P(C_i|D)\log_k(P(C_i|D))$$

Consider the following dataset, D, of training examples:

X_1	X_2	X_3	C
T	T	F	+
T	T	F	+
T	F	F	-
F	F	T	-
F	T	F	-
F	F	T	-

a) Compute $P(C_+|D)$ [2]

b) What is the entropy of this dataset with respect to the target attribute C? [3]

c) Write down a general expression for *information gain* in a decision split and then use it to compute the information gain of an X_3 decision split on D. [5]

Question 1 . . .

Marking schedule (all logs are in base 2)

a) $\frac{1}{3}$

b) $H(D) = \frac{1}{3}\log 3 + \frac{2}{3}\log\frac{3}{2}$

c) Define the weighted entropy of a decision split as:

$$H(D_L|D_R) = \frac{|D_L|}{|D|}H(D_L) + \frac{|D_R|}{|D|}H(D_R)$$

and then the **information gain** for the split is given by:

$$IG(D, D_L, D_R) = H(D) - H(D_L|D_R)$$

A split on X_3 will have weighted entropy:

$$\frac{1}{3}(0) + \frac{2}{3}(\frac{1}{2}\log 2 + \frac{1}{2}\log 2) = \frac{2}{3}$$

and so information gain in an X_3 split is:

$$\frac{1}{3}\log 3 + \frac{2}{3}\log\frac{3}{2} - \frac{2}{3} \approx 0.2516292$$

Question 2 (10 marks)

Consider a training set of n observations, $\{\vec{x}_i, y_i\}_{i=1\ldots n}$, where the $\vec{x}_i$ are 2 dimensional feature vectors and $y_i \in \{-1, +1\}$ is the target attribute that places each observation in one of two possible classes.

In linear classification, we seek to divide the two classes by a linear separator in the feature space.

a) Explain with the aid of a diagram, the role played by the parameters b and $\vec{w}$ for a linear separator. [2]

b) Prove that the decision boundary generated by $(\vec{w}, b)$ is identical to the decision boundary generated by $(s\vec{w}, sb)$ for any scaling parameter s. [2]

c) Assuming your data set is linearly separable, write down the perceptron algorithm, due to Rosenblat 1958, for finding a linear separator. [4]

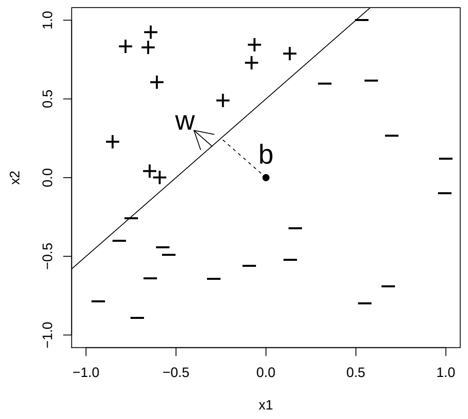
d) In 1962 Novikoff proved convergence of the perceptron algorithm under the following assumptions:

   Assumptions: Let $R = \max ||\vec{x_t}||$ and suppose that the learning task is solvable via a separator that passes through the origin. i.e. there exists some vector $\vec{w^*}$ of unit length and some $\delta > 0$ such that $Y_i(\vec{w^*} \cdot \vec{x_i}) > \delta$ for all t.

   Sketch a diagram showing separable data and an interpretation for R and δ. and write down an upper bound on the number of perceptron updates before convergence of the perceptron algorithm is guaranteed. [2]

Question 2 . . .

Marking schedule



a)

In the diagram feature vectors are two-dimensional. We wish to find a line that separates the positive observations form the negative ones. Any line can be specified by a unit normal vector, $\vec{w}$, and a signed perpendicular distance from the origin, b. Classification of a point $\vec{x}$ is then accomplished according to the value of $sign(\vec{x}\cdot\vec{w}-b)$. In the diagram above, a particular $\vec{w}$ and b is given that accomplishes the separation. The perceptron algorithm will discover such a separator. Note that in the above diagram b is negative with respect to the direction of $\vec{w}$.

b) Classification proceeds as follows:

if $x \cdot w + b > 0$, then x is positive, otherwise x is negative

now if $s > 0$, then

$x \cdot (sw) + (sb) = s(x \cdot w + b)$

which is > 0 if $x \cdot w + b > 0$.

therefore scaling by s of w and b does not change classification

Question 2 . . .

c)
```
perceptron = function(x, y, learning.rate=1) {
  w = vector(length = ncol(x)) # initialize w
  b = 0 # Initialize b
  k = 0 # count updates
  R = max(apply(x, 1, euclidean.norm))
  made.mistake = TRUE # to enter the while loop
  while (made.mistake) {
    made.mistake=FALSE # hopefully
    yc <- classify.linear(x,w,b)
    for (i in 1:nrow(x)) {
      if (y[i] != yc[i]) {
        w <- w + learning.rate * y[i]*x[i,]
        b <- b + learning.rate * y[i]*R^2
        k <- k+1
        made.mistake=TRUE
      }
    } }
  s = euclidean.norm(w)
  return(list(w=w/s,b=b/s,updates=k))
}
```

d) Under these assumptions, the perceptron algorithm converges after at most $(\frac{R}{\delta})^2$ updates.

Question 3 (10 marks)

Study the following clustering algorithm and the example data frame on the next page then answer the questions on the following page.

DIANA is a divisive clustering algorithm that proceeds by a series of successive splits. At step 0 all objects are together in a single cluster. At each step a cluster is divided, until at step $n-1$ all objects are apart (forming n clusters, each with a single object).

Each step divides a cluster, let us call it R into two clusters A and B say. Initially, A equals R and B is empty. In a first stage, we have to move one object from A to B. For each object i of A, we compute the average dissimilarity to all other objects of A:

$$d(i, A - \{i\}) = \frac{1}{|A|-1} \sum_{j \neq i} d(i,j)$$

The object $i^{'}$ for which the equation above attains its maximal value will be moved, so we put

$A_{new} = A_{old} - \{i^{'}\}$ and $B_{new} = B_{old} + \{i^{'}\}$

In the next stage we look for other points to move from A to B. As long as A still contains more than one object, we compute

$$d(i, A - \{i\}) - d(i, B) = \frac{1}{|A|-1} \sum_{j \neq i} d(i,j) - \frac{1}{|B|} \sum_{h \in B} d(i,h)$$

for each object i of A and we consider the object $i^{''}$ that maximises this quantity. When the maximal value of the equation above is strictly positive, we move $i^{''}$ from A to B and then look in the new A for another object that might be moved. On the other hand, when the maximal value of the difference is negative or 0 we stop the process and the division of R into A and B is completed.

At each step of the divisive DIANA algorithm we also have to decide which cluster to split. For this purpose we compute the diameter

$$diam(Q) = \max_{j \in Q, h \in Q} d(j,h)$$

for each cluster Q that is available after the previous step, and choose the cluster for which diameter is largest as the next cluster to split.

Question 3 continued ....

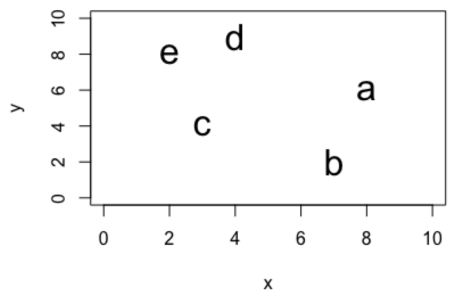
Consider the data frame:

```
x <- c(8,7,3,4,2)
y <- c(6,2,4,9,8)
( df <- data.frame(x, y, row.names=c("a","b","c","d","e") ) )
  x y
a 8 6
b 7 2
c 3 4
d 4 9
e 2 8
```

which when plotted using

```
plot(df,pch=row.names(df),xlim=c(0,10),ylim=c(0,10))
```

produces:



Now given that the *Manhattan* distances between objects can be computed in R using:

```
( dm <- as.matrix(dist(df,method="manhattan")) )

  a  b c  d  e
a 0  5 7  7  8
b 5  0 6  ? 11
c 7  6 0  6  5
d 7  ? 6  0  3
e 8 11 5  3  0
```

Question 3 continued ....

a) Explain how the *Manhatten* distances are calculated. [2]

b) Two entries in the distance matrix have been intentionally omitted. Complete the distance matrix by calculating the *Manhatten* distance from object `b` to object `d`. [1]

c) make use of the now completed matrix as the distance measure in the DIANA algorithm to cluster these **five** objects into **two** clusters. Show all your working. [7]

Question 3 . . .

Marking schedule

a) If x and y are d dimensional feature vectors, then

$$\text{Dist}_{\text{M}}(x, y) = \sum_{i=1,d} |x_i - y_i|$$

b)

$$\text{Dist}_{\text{M}}(b, d) = |7 - 4| + |2 - 9| = 10$$

c) Put all objects into class A and look for a *splinter* object, Compute row sums to get:

	a	b	c	d	e	T
a	0	5	7	7	8	27
b	5	0	6	10	11	32
c	7	6	0	6	5	24
d	7	10	6	0	3	26
e	8	11	5	3	0	27

and we see that object `b` is further from all other objects on average.

Put `b` into class B. and compute:

$$\text{Dist}_{\text{M}}(a, A) - \text{Dist}_{\text{M}}(a, B) = \frac{1}{3}(7 + 7 + 8) - 5 = 1\frac{1}{3}$$

which is strictly positive. If we do this for `c`, `d` or `e` we get negative results. So we move `a` to class B.

Repeat this computation to find that no more objects move. Thus the two classes are:

$A = \{$`c,b,e`$\}$ and $B = \{$`a,b`$\}$.

Question 4 (10 marks)

The R package, `tm`, provides a suite of routines for undertaking text mining tasks.

a) Describe the *corpus* data structure and outline 4 procedures that are commonly used for cleaning a corpus. [3]

b) What is a *term-document matrix* and what is it used for? [2]

c) Give a description of *sentiment analysis* and outline a text mining workflow for using Sentiment to enhance market prediction software. [5]

Question 4 . . .

Marking schedule

a) A corpus is a collection of documents. If `myCorpus` is a corpus containing n documents then each document can be accessed via integer indexing, `myCorpus[[i]]`, or named indexing `myCorpus[["ovid_2.txt"]]`. If the corpus is written to disc then all the documents are stored in a directory and each document name is used as that documents filename.

   Via the `tm_map` function, the tm packages provides common text cleaning operations:

   1) `tolower`, transform all characters to lower case.
   2) `stripWhitespace`, remove white space
   3) `removePunctuation`, get rid of punctuation characters
   4) `removeWords, myStopwords`, get rid of common words that may distort later analysis
   5) `gsub, pattern="search_string", replacement="repacement_string"`, homegrown cleaning

b) A term-document matrix represents the relationship between terms (or words) and documents, where each row stands for a term and each column for a document, and an entry is the number of occurrences of the term in the document. Correlations, associations, clustering and classification is usually carried out on the term document matrix and not the original corpus.

c) Sentiment analysis is the task of assigning sentiment valence values to documents. Valence could be binary or on some scale say 0-1 for example.

   To do this one would obtain a vocabulary, such as ANEW, where each of the words has already had a *sentiment valence* on a scale of 0-1 assigned.

   These valences may be assigned by human evaluations or learnt via machine learning from texts that have had valences assigned by humans.

   To estimate the overall valence of a text then one calculates a weighted average of the valence of the ANEW study words appearing in the text.

$$v_{text} = \frac{\sum_{i=1}^{n} v_i f_i}{\sum_{i=1}^{n} f_i}$$

   One can now envisage a software that looks for correlations between lagged tagged twitter feeds and market prices. A technique called *Granger Causality* has been employed to find lagged correlations in time series and papers have appeared in the literature to show that twitter sentiment can indeed predict market prices.

more space if you need it . . .