CCS Problem Set 1
Due: 4 April @ 11.55pm

Your writeups should include text and figures. Many problems can be solved either analytically or with code: unless specified otherwise, you may do whichever one you wish (and need not do both!). If you code something up, you should always include the (commented) code in your write-up in addition to reporting the solution. We have written the problem set assuming that you are coding in R; if you would like to use a different language, please contact Wai Keen first. Writeups should be submitted as pdfs; LaTeX is preferable but not obligatory.

In this problem set, Earth has just been invaded by evil aliens from another dimension. These aliens, the Klinklefigs, are a scary, tentacled lot who have descended on the planet with the purpose of destroying all that we hold dear (and eating our brains). You decide to use your new knowledge from CCS to help save the Earth from this disaster.

1. **Brain preferences among the Klinklefigs** (10 marks)

   One of your most pressing concerns is figuring out how likely it is for any given Klinklefig to want to eat *your* brain. Luckily, you and your fellow CCS friends have captured three Klinklefigs, which tend to spend a lot of their time hiding under the bed in their prison cell. You have named the three Klinklefigs Dan, Amy, and Wai Keen. The problem is, unlike the real Dan, Amy, and Wai Keen, it is very hard to tell the difference between these three. This matters because some of them seem more interested in eating your brain than others (you can tell a Klinklefig wants to eat your brain if it lunges at you saying "Yum!"). You therefore would like to be able to figure out which Klinklefig is which.

   You have noticed that Dan is quite shy, only appearing out from under the bed 20% of the time; Wai Keen and Amy appear out from under the bed 40% of the time each. It also appears that Wai Keen only wants to eat your brain 25% of the time, Amy wants to eat it about 30% of the time, and Dan has a voracious appetite for brains and wants to eat it 45% of the time.

   (a) What is the hypothesis space for this problem?

   (b) What is the prior distribution for this problem?

   (c) What are the likelihood functions for this problem?

   (d) You walk into the cell and see one Klinklefig out from under the bed. It lunges at you, saying "Yum!". It clearly wants to eat your brain. What is the probability that it is Dan?

2. **Klinklefig food choices** (10 marks)

   Over time, the number of captured Klinklefigs grows to nine. This enables you to investigate their food choices in a more general sense, not just restricted to the important issue of the tastiness of brains. After experimenting with many possible diets, you realise that Klinklefigs will only eat six possible earth foods: ants, brains, cucumbers, dandelions, emu, and feet. They like some of these foods more than others: you have observed that ants are only eaten 5% of the time, brains are eaten 40% of the time, cucumbers 10%, and so on. You have summarised these findings with the following snippet of R code. (Feel free to make use of this R code in the subsequent questions.)

```
#  ants brains cucumbers dandelions emu feet
pFood <- c(0.05,0.4,0.1,0.15,0.2,0.1)
```

   Eventually you succeed in brainwashing your Klinklefig prisoners to help you out. You therefore sit them all down and ask them to rate all six foods on a scale from 1 (awful) to 7 (magnificent). For instance, it turns out that 70% of Klinklefigs give ants a rating of 1 out of 7, 20% give it a rating of 2 out of 7, and a strange 10% give it a rating of 7 out of 7. You summarise the results for all food items in another snippet of R code:

```
pRating <- rbind(c(0.7,0.2,0.0,0.0,0.0,0.0,0.1), # Ants
                 c(0.1,0.1,0.0,0.0,0.0,0.7,0.1), # Brains
                 c(0.1,0.1,0.2,0.3,0.2,0.1,0.0), # Cucumbers
                 c(0.1,0.2,0.2,0.2,0.1,0.1,0.1), # Dandelions
                 c(0.1,0.1,0.2,0.3,0.1,0.1,0.1), # Emu
                 c(0.2,0.3,0.2,0.0,0.3,0.0,0.0)) # Feet
```

Each row in `pRating` corresponds to a food, and each column to a possible rating (column 1 corresponds to rating 1, column 2 to a rating of 2, and so on). To make it easier to read, here is the same information in a table:

	Rating						
Food	1	2	3	4	5	6	7
Ants	70%	20%	0%	0%	0%	0%	10%
Brains	10%	10%	0%	0%	0%	70%	10%
Cucumber	10%	10%	20%	30%	20%	10%	0%
Dandelions	10%	20%	20%	20%	10%	10%	10%
Emu	10%	10%	20%	30%	10%	10%	10%
Food	20%	30%	20%	0%	30%	0%	0%

(a) One day you overhear the Klinklefigs talking about the meal they were given that day (which consisted of one of the six food items). One of them says "Ugh. It was so horrible. I would only rate it as a 1." For each food, what is the probability that it was served that meal? In other words, what is your posterior distribution over food items given this data?

(b) The other Klinklefigs now all weigh in with their ratings of the meal. Here is their data:

klinklefig	a	b	c	d	e	f	g	h	i
rating	1	4	3	5	2	5	3	5	1

Note that "klinklefig a" is the same one who gave you the rating in part (a). The others are all different Klinklefigs. Now what does your posterior distribution over foods look like?

(c) Klinklefig b suddenly says "Wait! I changed my mind! I think I would give it a 2, not a 4." How does this change your posterior distribution? Why does it change so sharply?

3. **Learning to categorise zombies** (30 marks)

As is traditional for an alien invasion, when the Klinklefigs devour a human brain, the corpse rises from the dead as a zombie. Ghastly as this is, it has opened up a new applied problem that needs to be solved: learning to classify zombies. Our agents in the field have been able to capture and dissect a number of these zombies, and we have learned that they come in four basic types, unimaginatively named Zombie Type 1, 2, 3 and 4. They don't *look* much different to each other, but they seem to have different characteristics. Observationally, our field agents are able to assess zombies in terms of five characteristics: an estimate of the *health* of the zombie, an *armour* rating, a *speed*, an assessment of its *aggression* and its overall *damage* output.

You have recently been appointed as the resident Zombie Scientist, and have been asked to develop a tool that can infer the underlying type of a zombie based on the five observable ratings that a field agent provides. Your tool will be a k-nearest neighbours classifier, and your data can be found either in the `zombieSightings.Rdata` file or the `zombieSightings.csv` file released with this problem set.

For this exercise you can either use the `kNN` function in the `classifiers.R` file released in class, or you can write your own k-NN classifier if you prefer. But please note: if you're using the code from the `classifiers.R` file, **make sure you have the most recent version before you start the problem set!** The original release of this file had a tiny bug in the `kNN` function that I've now fixed. If you use an older version you'll get slightly incorrect results.

As for the data set, in the `zombieSightings.Rdata` file you'll find a single matrix called `zombies`. Each of the 300 rows corresponds to a single zombie sighting. Here's the first 8 rows:

```
> zombies[1:8,]
     health armour speed aggression damage type
[1,]     76   1194   235        134    267    1
[2,]    121   1260   569        131    238    1
[3,]    137   1278   108        131    301    3
[4,]    100   1155     1        115    208    4
[5,]    142   1221   236        125    226    1
[6,]    168   1242     1        130    299    2
[7,]    140   1114    63        103    158    2
[8,]    186   1297   511        131    287    3
```

As you can see, the first 5 columns are the *features*, and the 6th column is the *label*. The goal overall will be to use the feature information (i.e., `zombies[,1:5]`) to predict the label (i.e., `zombies[,6]`).

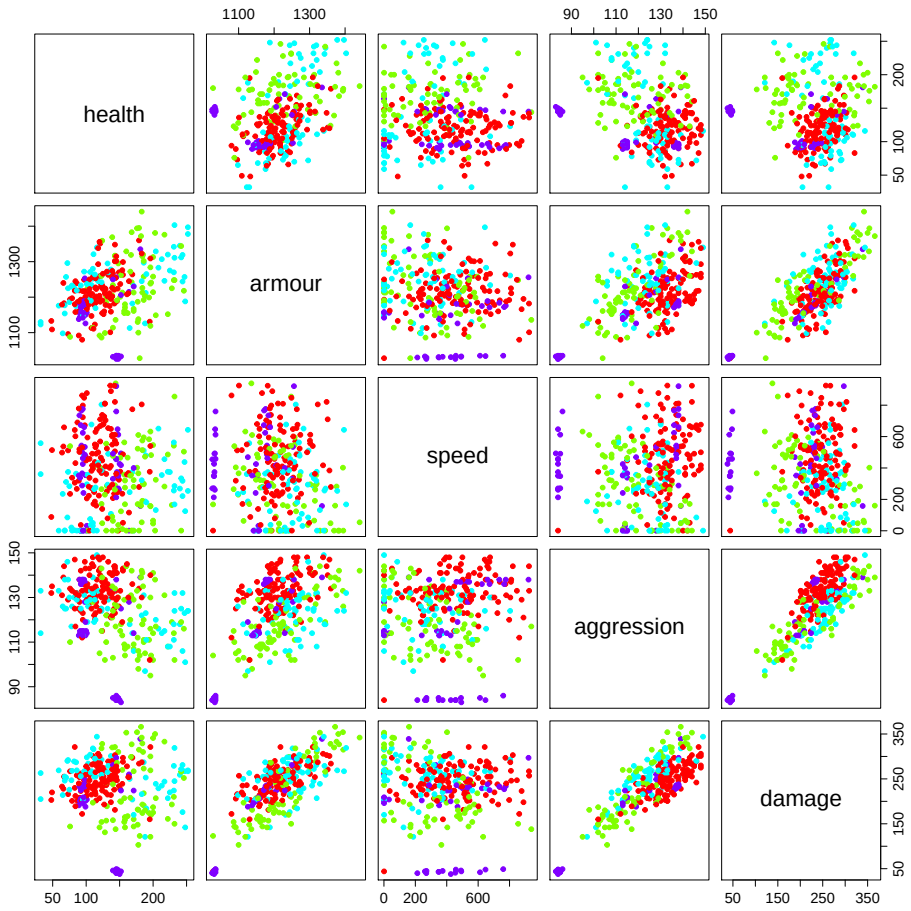

Figure 1: Scatter plots showing every pair of feature values. The red dots are Type 1 zombies, the green dots are Type 2 zombies, the light blue dots are Type 3 zombies and the purple dots are Type 4 zombies.

You can immediately tell that the data are going to be kind of weird-looking. Notice that some of the zombies have speed 1, whereas the others have quite large numbers. That's not an accident. Sometimes a zombie is so badly mangled that it can barely move, but most zombies are actually pretty quick. Clearly, we have messy data to work with! To give you a feel for the data set as a whole, Figure 1 draws a scatter plot showing every possible pair of features, plotting the four types of zombies in different colours.

(a) The first thing to do is get a classifier running on a small data set. Run a k nearest neighbours classifier using only the first 20 zombies (i.e., `zombies[1:20,]`) as the training data, and using the *last* 100 zombies (i.e., `zombies[201:300,]` as your test data. Run the classifier for all values of k from 1 to 10 nearest neighbours. For all values of k, what proportion of the zombies in the test set are correctly classified? How good is this classifier relative to what you'd expect by chance?

(b) Now that we've got a classifier working, let's give it a bit more data to work with. Repeat the exercise for part (a), but using the first 200 zombies as training data and the last 100 zombies as test data. Compare your new results to your previous ones. Was one version better than the other? If so, why?

(c) One possible problem with the exercises that we've just run is that we're using the raw data as our input, but all the features seem to be on different scales. The table below shows you the maximum value and the minimum value for each of the different features, along with the difference between the two (i.e., the range):

	health	armour	speed	aggression	damage
minimum	32	1026	1	83	38
maximum	252	1441	939	149	366
range	220	415	938	66	328

It is sometimes a good idea to try rescaling the raw data. Specifically, rescale all of the features (not the label!) so that the minimum value is 0 and the maximum value is 1. Include the rescaled feature values for the first 8 zombies in your answer (similar to the way I did with the raw data at the start of this question). For the rest of this question, we'll work with the rescaled data and not with the raw data.

(d) Repeat the exercise from part (b) using the rescaled data. That is, run a k-nearest neighbours classifier for all values of k between 1 and 10, using the first 200 observations as training data and the last 100 as test data. Does the classifier work better on the rescaled data?

(e) Why is there a difference between the results in parts (b) and (d)? For this question you don't need to prove anything or do any calculations. Just describe in words *why* you think the classifier behaves differently when you rescale the data.

(f) The overall accuracy can be a bit misleading. Not every category is guessed correctly with the same effectiveness. Repeat the exercise from part (d), but this time report the how accurately each of the zombie types is classified. For example, if there are 20 Type 1 zombies in the test set, how often are those correctly classified as Type 1 zombies? Do this for all four Types, and for all values of k between 1 and 10.

(g) With respect to the results in part (f), take a close look at the pattern of accuracy for Type 1 zombies as k increases. Compare that to the pattern for Type 4 zombies. Are they the same or different? Try to explain why you're seeing the pattern that you do. Hint: try calculating how many of each type of zombie exist in the data. What does that mean when you calculate k nearest neighbours for different values of k for those types?

(h) When training classifiers in real life, it's not a good idea to rely solely on a single train-test split. A better approach is to randomly divide the data into training and test sets several times (with each time containing different training and testing data), and then average the results. That's what we'll cover in this exercise. You'll need to randomly divide the data into 10 train-test splits (the `sample` function in R may be helpful to you in doing this). Each training set should contain 200 observations and each test set should contain 100 observations. How well does the classifier perform, for all values of k between 1 and 20? In your answer you should report the overall accuracy as well as the category-specific performance in (f).

4. **Sub-types of Klinklefigs** (20 marks)

After extensive experience with Klinklefigs, you realise that there might be several different races. Although it's hard to tell the difference between Klinklefigs, you have been able to identify two dimensions that appear to differentiate: WRINKLINESS and HEIGHT. You send many scouts out to gather data about the distribution of these features in the population: for every Klinklefig they see, they rate its WRINKLINESS and measure its HEIGHT. This is shown in Figure 2 and the information can be found in the file `klinklefigs.RData` available with this problem set.

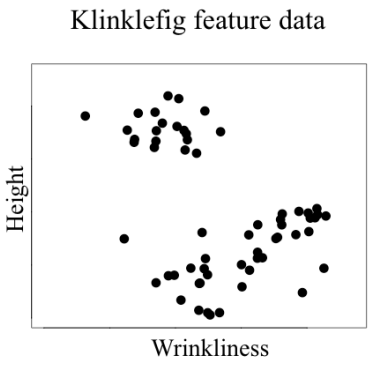

Figure 2: A graph of all of the klinklefigs your scouts have classified.

(a) Download the code called `mixtureofgaussians.R`, which corresponds to the code presented in lecture. It is complete except for the line(s) of code that perform the E-step and M-step; locations where this code should go are clearly specified in the file. Fill in the missing code so that it now fully works. (Hint: you may use – but don't have to use – the function `getgaussiandistance`, which is in the `kmeanshelperfns.R` file). Submit the code corresponding to the modifications you made.

(b) Based on this data, you want to figure out how many distinct races of Klinklefigs there are. To do this, we'll have you run your Mixture of Gaussians (MoG) code on this dataset many times for different possible numbers of clusters k, and for each k calculate the amount of agreement between different solutions. The file `mogsupportfns.R` contains a function called `getrandindex` which will allow you to calculate the overlap between any two different solutions based on a measure called Adjusted Rand Index (*adjRand*). It takes as arguments two r_{curr} matrices (from two different runs of `mixtureofgaussians`) and returns a value that is 1 if the matrices/clusters are identical, 0 if they are no more similar than would be expected by chance, and mid-range values to reflect some degree of overlap.

Run your MoG code 10 times each for $k = 2$, $k = 3$, and $k = 4$. Then use `getrandindex` to calculate the *adjRand* for every possible pairing of datasets for each k. That is, you'll be comparing all of your results for $k = 2$ to each other, all your results at $k = 3$ to each other, and so on; this will be 45 pairs for each k. You need not compare results from $k = 2$ to $k = 3$ or $k = 4$. Doing this will yield a measure of the amount of agreement or similarity between different runs of `mixtureofgaussians`.

What is the mean *adjRand* at each k? The standard deviation? (You can use the `mean` and `sd` function for this).

(c) What do you think your result in (b) reflects about how many distinct races there really are? In general, do you think estimating the "true" value of k by looking at the amount of agreement between different runs makes sense? Why or why not? (Note: you do not need to have succeeded in coding the `mixtureofgaussians` in order to answer this "in general" question; it is a thought question only).

(d) Another idea for determining how many races there really are is to compare the likelihoods for each of the best clusterings found for each possible number of races. I did this for 2, 3, and 4 races and found that two races had the lowest likelihood, three races the next highest, and four races the highest likelihood. Does this suggest that there are actually four distinct races? Why or why not? (Note: you do not need to have succeeded in coding the `mixtureofgaussians` in order to answer this question; it is a thought question only).