The problem sets in this class will be done using Matlab. You may need to use the computers in the EECS labs to get access to Matlab. If you have problems accessing these computers, contact the GSIs at `gsi-cogsci131@lists.berkeley.edu`.

Collaboration Policy

Class policy is that collaboration is permitted when solving problems, but each student must: (1) write his or her own code; (2) write the document containing his or her answers independently; and (3) list the people he or she worked with on his or her problem set.

Submission Instructions

Problems will require you to either turn in Matlab code, written answers, or both. All of the files you need are provided for you – you just need to fill them in with your answers. **Make sure to fill in your name and collaborators for each part of the homework.**

All parts that include Matlab code come with a validation script called `validatePN.m` which you can run by calling `validatePN()` from Matlab (replace `N` with the problem number, e.g. for Problem 1, run `validateP1()`). Make sure you run these functions for each part of the homework before submitting your assignment. **If your code does not pass the validation script, you will NOT receive credit.** Note, however, that this is not a grading script; it is entirely possible for incorrect answers to pass the validation script.

You will be submitting your homework on bSpace. For each problem set, you will submit a single zip file containing your filled-in files from each problem. The directory structure within your submission should be the same as when the problem set was released. The zip file should be named:

```
YourLastName_YourFirstName_ProblemSetNumber.zip
```

For example, if your name is Jane Doe, your submitted file should be named `Doe_Jane_N.zip`, where `N` is the problem set number. You can update your submission until the due date. If you make any changes to your submission after the due date, we will regard your problem set as having been turned in at the last time it was changed.

If you use Matlab on the computers in Cory 105 or log in with remote desktop, please note that files saved to the Desktop are removed when you log out. Instead, save any files to your user directory, `U:/`. The `U:/` drive can also be accessed through a shortcut on the Desktop.

Late Problem Set Policy

This problem set is due on **March 20, 2014 at 2:00pm**. You get **THREE LATE DAYS** total for the entire semester, to account for foreseeable minor crises. If you turn in a problem set late after using these late days, 20% of the total points on the problem set will be deducted for each 24 hours (or portion thereof) that it is late.

1 Multidimensional scaling (3 points)

Multidimensional scaling (MDS) is a statistical technique that has been used by cognitive scientists to create spatial representations of the similarities among a set of stimuli. Matlab has several commands that perform different types of multidimensional scaling. In this problem we will use nonmetric multidimensional scaling, which finds a spatial representation where the *ordering* of distances between points in $\mathbb{R}^n$ corresponds to the ordering of dissimilarities among a set of stimuli.

To begin, load `kinship.mat` from `ps03/problem1/kinship.mat` using the `load` command. This should load two variables into the workspace:

- `names`: A 12×1 cell array containing the names of 12 kinship terms
- `similarities`: A 12×12 matrix where index (i, j) lists the similarity between the kinship term in `names{i}` and the kinship term in `names{j}` as rated by human subjects. Note that the similarity scores in `similarities` have been scaled to fall between 0 and 1.

In Parts A, B, and C you will be completing the function `nonMetricMDS` which takes `names` and `similarities` as arguments and outputs `sim2D` and `plot1` (described below).

Part A

The Matlab command `mdscale` performs nonmetric multidimensional scaling (use `doc mdscale` to learn more about its operation). We wish to use nonmetric MDS to make a two dimensional spatial representation of the similarities between kinship terms.

For Part A call `mdscale` within `nonMetricMDS` so that it takes `similarities` as an input and returns `sim2d`, a 12×2 matrix containing x and y coordinates for the 12 kinship terms in $\mathbb{R}^2$. This means that within `nonMetricMDS` you should have a line of the form:

```
sim2d = mdscale(arg1, arg2, ...);
```

where `arg1, arg2, ...` are placeholders for the arguments you will pass to `mdscale`. The number of arguments you'll need to complete this part may differ from the number listed in the example above.

Part B

Next you will create a scatter plot for the points in `sim2d`. Within `nonMetricMDS` use the `plot` command to create a plot with blue **square** at the location of each of the kinship terms (the documentation for `plot` (see `doc plot`) may be helpful for learning how to make it generate what you want). Name your plot object `kinshipPlot`. This means that within `nonMetricMDS` you should have a line of the form:

```
kinshipPlot = plot(arg1, arg2, ...);
```

where `arg1, arg2, ...` are placeholders for the arguments you should pass to `plot`. Note that again, the number of arguments you'll need may differ from the number listed in the example.

Part C

Finally, you will use the `text` command to add labels to the points you plotted in `kinshipPlot`. Find the line in `nonMetricMDS` with the command `hold on;` – this tells Matlab that you will be adding information to the graph, and to not remove the points that you have already plotted (`clf` clears the figure). Below this line use the `text` command (see `doc text` for usage) within a simple `for` loop to set the kinship terms in `names` as labels for the points in `kinshipPlot`. Save each text object within the structure array `labels`. That is, for Part C you should have code that resembles:

```
for i=1:length(similarities),
    labels(i) = text(arg1, arg2, ...);
end
```

where `arg1, arg2, ...` are placeholders for the arguments you should pass to `text`. Note that again, the number of arguments you'll need may differ from the number listed in the example.

Save your MDS plot as as a JPEG named `mds1.jpg` to `ps03/problem1` using the File → Save As menu in the figure window.

Part D

Using the provided written answers template `writtenAnswersPartD.txt`, please answer the following question:

How well does the spatial representation produced by `mdscale` capture your intuitions about the similarities between these kinship relationships? Justify your answer.

2 Hierarchical clustering (2 points)

An alternative to multidimensional scaling is hierarchical clustering. Helpfully, commands for hierarchical clustering are built into Matlab. For this problem you will be using the same `kinship.mat` dataset that you used in Problem 1. Load its contents to the workspace by following the instructions in Problem 1.

Part A

We wish to create a hierarchical clustering for the *dissimilarities* between the terms in `names` using the similarity data from `similarities`. Before we can do this, however, we must translate our data into a format that Matlab's `dendrogram` function recognizes.

First we must convert our similarity matrix (`similarities`) into a dissimilarity matrix. One simple way to do this is to subtract each item in `similarities` from 1. This means that the contents of cell $[i, j]$ in our dissimilarity matrix will be equal to 1 minus the contents of cell $[i, j]$ in our original similarity matrix. Save this new dissimilarity matrix as the variable `disSim` in `makeDendrogram`. Note that `size(similarity)` should equal `size(disSim)`).

Hint: This can be done without using a `for` loop.

Part B

We need to further modify our dissimilarity data before we can pass it to `dendrogram`. Use the `squareform` function on `disSim` to convert `disSim` from a square matrix into a row vector containing only the elements below the diagonal (this is okay to do since `similarities` and `disSim` are *symmetric* along their diagonal). The result of this operation should be a 1×66 row vector. Save this vector as `lowerDiag`.

Finally, we must convert our new vector into a linkage matrix. This is done by passing it as an argument to the `linkage` function in Matlab. If you are interested in understanding what the linkage function is doing, try looking at `doc linkage` or asking one of the GSI's. The output from this operation should be an $(n - 1) \times 3$ real-valued matrix. Save it as `linkageMatrix` in `makeDendrogram`.

Part C

Finally we're ready to create our dendrogram. After completing Parts A and B, use

```
dgram = dendrogram(linkageMatrix,'LABELS', names);
```

to define the dendrogram of dissimilarities between the kinship terms in `names`. Once you have added this to the end of your function `makeDendrogram`, try running it to verify that it produces a dendrogram.

In the dendrogram the names of the 12 kinship terms should appear at the bottom, with individual branches merging higher up to indicate different clusters. The vertical axis in the plot measures the dissimilarity between the branches being merged. Since we are using dissimilarities, the lower on the tree that two branches merge, the *more similar* the two branches are to one another.

Save your dendrogram plot as as a JPEG named `den2.jpg` using the File → Save As menu in the figure.

Part D

Using the provided written answers template `writtenAnswersPartD.txt`, please answer the following question:

How well do the results of the hierarchical clustering capture your intuitions about the similarity of the kinship terms? Justify your answer.

3 Spaces and features (5 points)

In this problem, we will revisit non-metric MDS and hierarchical clustering on a different dataset. Load the file "eurodist.mat". This file contains two two variables,

- `city_names` – A 1×21 cell array containing the names of 21 European cities
- `city_dist` – A 21×21 distance matrix where cell $[i, j]$ contains the distance between `names{i}` and `names{j}`

In Part A and B you will complete the function `scalingVsClustering`, to run non-metric MDS and hierarchical clustering analyses on this data.

Part A

Using problem 1 as a guide, perform a non-metric multidimensional scaling analysis to obtain a 2-dimensional representation of the distance data in `city_dist`. Save the `mdscale` results to the variable `mdsCoordinates` and use the `plot` and `text` functions to plot and label your points. Make each data point in your plot a red **pentagram**. Save your plot as the variable `mdsPlot`. Save your labels in the structure `labelMDS`.

Save this mds plot as as a JPEG named `mds3.jpg` using the File → Save As menu in the figure window.

Note: Because we are using *non-metric* MDS to create our representation, the absolute position of any point in the plot is not meaningful in and of itself. Why is this?

Part B

Using problem 2 as a guide, perform hierarchical clustering on the `eurodist.mat` dataset to produce a dendrogram with labels. As before, save the output of the dendrogram function call to the variable `dendPlot`.

Save this dendrogram plot as as a JPEG named `den3.jpg` using the File → Save As menu in the figure.

Note: Because we are using metric distances and not similarities in this problem you will not be able to use the exact code you used in problem 2. Think about why we calculated dissimilarities in problem 2 and how the magnitude of metric distances in `city_dist` relate to the (dis)similarity ratings used previously.

Part C

Using the provided written answers template `writtenAnswersPartC.txt`, please answer the following question:

Was there a difference in which kind of representation seemed to capture your intuitions about the similarity of kinship terms from questions 1 and 2, and the pairwise distances of European cities? Why do you think this is? Is there a property of the two domains from which the stimuli are drawn that makes them more amenable to one representation than another?

Part D

Using the provided written answers template `writtenAnswersPartD.txt`, please answer the following question:

Name two more kinds of stimuli that you think would be represented best using hierarchical clustering, and two more kinds of stimuli that you think would be best captured by a spatial representation. Justify your answers.

4 Context model (4 points)

The context model, described by Medin and Schaffer (1978), is an exemplar model: the probability that a stimulus is assigned to a category is based on its similarity to all of the exemplars in that category. Let y denote a stimulus in memory, and x be a new stimulus. Each stimulus has m binary features. Assuming that there are no biases associated with the categories (the β_A and β_B terms you might remember from class), the probability that x is assigned to category A (as opposed to category B) is

$$P(A|x) = \frac{\sum_{y \in A} \eta_{xy}}{\sum_{y \in A} \eta_{xy} + \sum_{y \in B} \eta_{xy}} \tag{1}$$

where $\sum_{y \in A} \eta_{xy}$ is the sum over all exemplars y in category A of the similarity of x to y. The similarity between x and a stored exemplar y is given by

$$\eta_{xy} = \prod_{k=1}^{m} \eta_{xyk} \tag{2}$$

$$\eta_{xyk} = \begin{cases} 1 & \text{if } x_k = y_k \\ s & \text{if } x_k \neq y_k \end{cases} \tag{3}$$

where $\prod_{k=1}^{m}$ indicates the product over the m individual features, x_k and y_k are the values of x and y on feature k, and s is some constant value that is a parameter of the model. This is a simplification of the equation given in lecture where s could differ based on which feature we were comparing; we assume all features have the same $s_k = s$.

We can rewrite this more compactly as:

$$\eta_{xy} = \prod_{k=1}^{m} s^{I(x_k \neq y_k)} \tag{4}$$

where $I(\cdot)$ takes the value 1 when its argument is true and 0 otherwise. Note that this reformulation works because if $I(x_k \neq y_k) = 1$, then the term for the kth feature is $s^1 = s$, and if $I(x_k \neq y_k) = 0$ (i.e., $x_k = y_k$), then the term for the kth feature is $s^0 = 1$.

Here are a few further details to help you understand Equation 4:

- If you have not seen Π before, it is just like Σ except you multiply each term together instead of sum. For example, $\prod_{k=1}^{5} k = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 5! = 120$. Or, if $\vec{v} = (4, 1, 2)$ then, $\prod_{k=1}^{3} v_k = 4 \cdot 1 \cdot 2 = 8$.
- Also, if you have not seen $I(\cdot)$ before, here are some examples of how it works: $I(3 = 3) = 1$, $I(3 = 4) = 0$, $I(3 \neq 3) = 0$, and $I(3 \neq 4) = 1$.
- It may help to think of Equation 4 as s^C, where C is the number of features for which x and y do not have the same value. If you are having trouble with how to code Equation 4, or figuring out what it does, try seeing what it would return by hand for a few values (like $\vec{x} = [1, 1, 1, 1]$ and $\vec{y} = [1, 1, 1, 1]$ or $\vec{y} = [1, 1, 0, 1]$ or $\vec{y} = [1, 0, 0, 1]$).

Part A

Using the provided function template `problem456/calculateSimilarity.m`, write code that calculates the similarity η_{xy} between two stimuli (as defined in Equation 4). When you run your function, the output should look something like this:

```
>> calculateSimilarity([0 0 1 1], [1 1 1 1])

ans =

    0.0100

>> calculateSimilarity([0 0 1 1], [1 1 1 1], 0.2)

ans =

    0.0400
```

Part B

Using the `calculateSimilarity` function you just wrote, write code that implements a version of the context model (i.e., computes Equation 1). Fill in the provided function template `problem456/contextModel.m` with your solution.

The function `contextModel` should have three input parameters: the features of the exemplars, the categories of the exemplars (organized as a vector where the i^{th} entry corresponds to the category of the i^{th} exemplar), and the features of the "test" stimuli for which you would like to compute category predictions. It should return a vector which has $P(A|x)$ for each stimulus x in the "test" stimuli, corresponding to the probability that each test stimulus x is in category A.

Note: We assume that exemplars are stimuli for which you know the category labels (i.e., the stimuli "in memory"), and test stimuli are stimuli for which you do not necessarily know the correct category labels - that's why the function takes the categories of the training stimuli as input, and does not take the categories of the test stimuli as input. That means that when you compute $P(A|x)$ for some stimulus x, you will be using its similarity to the exemplars in the training set, but not to the exemplars in the test set.

When you run your function, the output should look something like this:

```
>> contextModel([[0 1 0 1]; [1 0 1 0]], [1 2], [[0 1 0 0]])

ans =
```

```
    0.9901

>> contextModel([[0 1 0 1]; [1 0 1 0]], [1 2], [[0 1 0 0]], 0.2)

ans =

    0.9615
```

Part C

Load `problem456/stimuli.mat`. This dataset contains some hypothetical stimuli, as follows:

- `exemplars` – a 6×4 binary matrix, where each row corresponds to an exemplar stimulus, and each column corresponds to a feature. If the element at location (i, j) is 1, then that means stimulus i has feature j; if it is 0, then stimulus i does not have feature j.
- `exemplarCategories` – a 6×1 vector indicating which category each training stimulus belongs to. If stimulus i belongs to category A, the the value of `exemplarCategories` at index i will be a 1; if it belongs to category B, the value will be a 2.
- `novel` – a 2×4 binary matrix (similar in format to `exemplars`). Each row corresponds to a novel stimulus, and each column corresponds to a feature.
- `novelCategories` – a 2×1 vector indicating which category each novel stimulus belongs to. Like `exemplarCategories`, a 1 indicates category A and a 2 indicates category B.
- `prototypes` – a 2×1 vector of category prototypes. The first row is the prorotype for category A, and the second row is the prototype for category B.

What does the function return for $P(A|x)$ when you use `exemplars` as the exemplars and `exemplars` as the test stimuli? You should call your function like this:

```
>> contextModel(exemplars, exemplarCategories, exemplars)
```

For each of the stimuli in `exemplars`, describe how it is classified by the context model (e.g., "The context model says that stimulus (insert number) is (insert relation) likely to be a member of category A than category B"). How do these classifications compare to the true categories (found in `exemplarCategories`)?

Put your answers in the file `problem456/writtenAnswersProblem4PartC.txt`.

Part D

What does the function return for $P(A|x)$ when you use `exemplars` as the exemplars and `prototypes` as the test stimuli? You should call your function like this:

```
>> contextModel(exemplars, exemplarCategories, prototypes)
```

For each of the stimuli in `prototypes`, describe how it is classified by the context model (e.g., "The context model says that stimulus (insert number) is (insert relation) likely to be a member of category A than category B"). How do these classifications compare to the true categories of the prototypes?

Put your answer in the file `problem456/writtenAnswersProblem4PartD.txt`.

Part E

What does the function return for $P(A|x)$ when you use `exemplars` as the exemplars and `novel` as the test stimuli? You should call your function like this:

```
>> contextModel(exemplars, exemplarCategories, novel)
```

For each of the stimuli in `novel`, describe how it is classified by the context model (e.g., "The context model says that stimulus (insert number) is (insert relation) likely to be a member of category A than category B"). How do these classifications compare to the true categories (found in `novelCategories`)?

Put your answer in the file `problem456/writtenAnswersProblem4PartE.txt`.

5 Prototype model (4 points)

A prototype model probabilistically classifies stimuli to the category that has the nearest prototype. For example, the categories A and B in the category structure in `stimuli.mat` have prototypes of $\mu_A = (0, 0, 0, 0)$ and $\mu_B = (1, 1, 1, 1)$. Assuming that there are no biases associated with the categories (the β_A and β_B terms you might remember from class), the probability of selecting category A for a stimulus x is given by

$$P(A|x) = \frac{\eta_{xA}}{\eta_{xA} + \eta_{xB}} \tag{5}$$

where η_{xA} is the similarity of x to the prototype μ_A, computed as in Equation 4.

Part A

Using the provided function template `problem456/prototypeModel.m`, write code that computes $P(A|x)$ for the prototype model. You may find it useful to use the `calculateSimilarity` function you defined in Problem 4A.

The function `prototypeModel.m` takes the prototypes as a matrix, where the first row is the prototype for category A and the second row is the prototype for category B. The model also takes in a matrix `testStimuli` that has the features of your stimuli. Your function should return a vector containing entries $P(A|x)$ corresponding to the probability that each stimulus x is in category A.

When you run your function, the output should look something like this:

```
>> prototypeModel([[0 0 0 0]; [1 1 1 1]], [[0 1 0 0]])

ans =

    0.9901

>> prototypeModel([[0 0 0 0]; [1 1 1 1]], [[0 1 0 0]], 0.2)

ans =

    0.9615
```

Part B

Load `problem456/stimuli.m` again. (Remember that the variables in this dataset are described above in Problem 4, Part C).

What does the function return for $P(A|x)$ when you use `prototypes` for the prototypes and `exemplars` as the test stimuli? You should call your function like this:

```
>> prototypeModel(prototypes, exemplars)
```

For each of the stimuli in `exemplars`, describe how it is classified by the prototype model (e.g., "The prototype model says that stimulus (insert number) is (insert relation) likely to be a member of category A than category B"). How do these classifications compare to the true categories (found in `exemplarCategories`)?

Put your answer in the file `problem456/writtenAnswersProblem5PartB.txt`.

Part C

What does the function return for $P(A|x)$ when you use `prototypes` for the prototypes and `prototypes` as the test stimuli? You should call your function like this:

```
>> prototypeModel(prototypes, prototypes)
```

For each of the stimuli in `prototypes`, describe how it is classified by the prototype model (e.g., "The prototype model says that stimulus (insert number) is (insert relation) likely to be a member of category A than category B"). How do these classifications compare to the true categories of the prototypes?

Put your answers in the file `problem456/writtenAnswersProblem5PartC.txt`.

Part D

What does the function return for $P(A|x)$ when you use `prototypes` for the prototypes and `novel` as the test stimuli? You should call your function like this:

```
>> prototypeModel(prototypes, novel)
```

For each of the stimuli in `novel`, describe how it is classified by the prototype model (e.g., "The prototype model says that stimulus (insert number) is (insert relation) likely to be a member of category A than category B"). How do these classifications compare to the true categories (found in `novelCategories`)?

Put your answers in the file `problem456/writtenAnswersProblem5PartD.txt`.

6 Model comparison (2 points)

You should now be able to generate predictions for $P(A|x)$ for all stimuli using either the context model or the prototype model.

Part A

For each of the *exemplars*, *prototypes*, and *novel* stimuli, which model gives a more accurate classification? For example, give us a sentence like "The context model is more/less/equally accurate than the prototype model for the (insert set) stimuli".

Put your answer in the file `problem456/writtenAnswersProblem6PartA.txt`.

Part B

Explain why each model shows its particular pattern of performance, and interpret whether each model shows a "prototype effect," in which it is better at classifying the prototype stimuli than other stimuli to which it hasn't previously been exposed.

Put your answer in the file `problem456/writtenAnswersProblem6PartB.txt`.