Computational Cognitive Science (2021–2022)

CCS Assignment 2021-2022

Submission Instructions

You should answer the questions in this Rmd file, and 'Knit' it into a PDF file, which you will submit. Upload your answer file on Blackboard Learn. Please make sure you have submitted the right file by downloading and reviewing the file you have submitted. We cannot make concessions for students who turn in incomplete or incorrect files by accident.

Good Scholarly Practice

Please remember University policies regarding all assessed work for credit. Details and advice about this can be found here.

You are expected to complete this assignment *individually* and must take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (generally permitting access only to yourself, or your group in the case of group practicals). Do not disclose your solutions when asking questions in a forum visible to other students (e.g., non-private questions on Piazza). When in doubt, contact course staff by email or private questions.

General Guidance

The assignment is out of 100. Marks are given alongside each question. Especially challenging questions are marked 'Stretch.' We recommend you leave them to the end if you are stuck on them and answer easier questions first.

Objective and materials

This assignment is designed to introduce you to simple models of causal learning, and assess your ability to (1) understand and implement computational models, and (2) evaluate and compare them using experimental data.

We will look at three models: RW, ΔP, and Causal Support. Our data are from real-world experiments where people made judgments about whether objects were 'blickets,' having seen events involving those objects. The data files you will need are `org-kva-adult-E1-OR.csv` and `org-kva-adult-E1-AND.csv`, contained in the `data` sub-directory in the same archive that contains this Rmd file.

You will need to install RStan to complete this assignment. You can find installation instructions here.

Background: The experimental setup

"Blicket machine" experiments use a fictional device called a "blicket detector" or "blicket machine" that reacts to "blickets" by activating (lighting up and playing a sound). That is, blickets are objects that can cause the blicket machine to activate. By showing people different events involving the machine and prospective blickets, experimenters can learn about how people reason about cause and effect.

Please have a look at (Gopnik et al. 2004)(link) for a description of blicket detector experiments and data like those we will be modelling.

Example: Backward blocking

Here is an example of a blicket detector experiment, similar to experiments that have demonstrated the *backward blocking* phenomenon.

The experimenter shows participants two objects, **A** and **B**, and says that one, both, or neither might be blickets.

They ask participants to judge whether **A** and **B** are blickets (0 for yes, 1 for no).

The experimenter then places **A** and **B** on the machine together. The machine activates, lighting up and playing a sound.

The experimenter asks for another judgment about how likely it is that **A** and **B** are blickets.

The experimenter then places **A** placed on the machine by itself. The machine activates again.

The experimenter asks for a final set of judgments about how likely it is that **A** and **B** are blickets.

When people see the final event, they often revise their beliefs about not just **A**, but about **B** as well. This phenomenon is called "backward blocking," and has been studied extensively with blicket detector experiments, as well as in earlier research that predates blicket detector experiments.

We can summarise the events participants see in the experiment above as "AB+ A+," where events are separated by spaces, letters denote the objects that are placed on the machine, and "+" and "-" indicate whether the machine activated.

Experiments

We will be considering results from previous experiments that are slightly more complicated than the above example. These experiments were not designed to distinguish between the models you will be comparing, but may nonetheless be useful in understanding differences between the models.

The table below summarizes them, using the same notation as above.

Condition Name	Abbrev	Event Set 1	Event Set 2
Conjunctive	CON	A- B- C- AB- AC+ BC-	D- D- D- E- DF+ DF+
Disjunctive	DIS	A+ B- C+ AB+ AC+ BC+	D- D- D- E- DF+ DF+

The experiment worked as follows: Participants were randomly allocated to either the CON or DIS condition, were then shown Event Set 1 for the respective condition, and subsequently made judgments about objects A, B, and C. Then participants were presented with Event Set 2 from the condition they were allocated to, after which they made judgments but this time about objects D, E, and F.

The data

There are two kinds of data we need to consider. The first is the data that participants see – the models will need to be provided those events in order to make predictions.

Event representation

Let us represent the sequence of objects placed on the machine as a (E, O) matrix where E stands for the number of events and O to the number of objects. Thus, each row describes the objects that are present in a single event, where a 1 entry indicates that the corresponding object is present, and a 0 entry indicating that the object is absent (with the columns ordered alphabetically). For the backwards-blocking example, this would be:

```r
bb_objects <- matrix(c(1,1,1,0),nrow=2,ncol=2)
print(bb_objects)
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    0
```

So the first row (1,1) means that in the first event, both objects are present on the machine.

We can use a vector of length E containing ones and zeros to represent whether the machine activates (1) or doesn't (0). For the backward blocking example above, the vector would thus be:

```r
bb_activations <- c(1,1)
```

This means that the machine activates for the first event, and also for the second event.

For the experiment we are looking at, there are more than two events and two objects to keep track of. Fortunately, the objects that are placed on the machine are the same in both conditions (CON and DIS), so we can use one matrix for both conditions:

```r
all_objects <- matrix(c(1,0,0,1,1,0,0,0,0,0,0,0,
                        0,1,0,1,0,1,0,0,0,0,0,0,
                        0,0,1,0,1,1,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,1,1,0,1,1,
                        0,0,0,0,0,0,0,0,0,1,0,0,
                        0,0,0,0,0,0,0,0,0,0,1,1),
                      ncol=6,nrow=12)
print(all_objects)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6]
##  [1,]    1    0    0    0    0    0
##  [2,]    0    1    0    0    0    0
##  [3,]    0    0    1    0    0    0
##  [4,]    1    1    0    0    0    0
##  [5,]    1    0    1    0    0    0
##  [6,]    0    1    1    0    0    0
##  [7,]    0    0    0    1    0    0
##  [8,]    0    0    0    1    0    0
##  [9,]    0    0    0    1    0    0
## [10,]    0    0    0    0    1    0
## [11,]    0    0    0    1    0    1
## [12,]    0    0    0    1    0    1
```

Make sure you understand how the rows of the matrix above correspond to `A B C AB AC BC D D D E DF DF`.

The difference in the conditions is in the activation of the machine:

```r
con_activations <- c(0,0,0,0,1,0,0,0,0,0,1,1)
dis_activations <- c(1,0,1,1,1,1,0,0,0,0,1,1)
```

Now that we have the evidence that people are being given in the experiment, we can also run this evidence through different computational models and see what predictions they make about human judgments. This allows us to assess how different models expect people will learn and generalize from that evidence, and since we have experimental data from actual people, we can compare model predictions to human judgments. We will go through an example of a computational model in the next section. Before that, let's look at some data from the experimental participants.

There are many columns in our CSVs, but the relevant ones for us are the binary judgments about whether objects A-F are blickets in both conditions. Note that the original experiment had people make two sets of judgments with different sets of objects, but here we will simply skip the second set.

```r
get_judgments <- function(path_to_csv) {
  human_data <- read.csv(path_to_csv)
  select(human_data,matches("isBlick.1"))
}

human_dis_data <- get_judgments("org-kva-adult-E1-OR.csv")
human_con_data <- get_judgments("org-kva-adult-E1-AND.csv")
```

The columns are `isBlickA1` through `isBlickF1`. We aren't using the second judgments here, so we're dropping `isBlickA2` and so on.

Here are mean judgments over all participants, separate for each condition:

```
colMeans(human_dis_data)
```

```
## isBlickA1 isBlickB1 isBlickC1 isBlickD1 isBlickE1 isBlickF1
## 0.9818182 0.1272727 0.8363636 0.1090909 0.1636364 0.9636364
```

```
colMeans(human_con_data)
```

```
##  isBlickA1  isBlickB1  isBlickC1  isBlickD1  isBlickE1  isBlickF1
## 0.31707317 0.03658537 0.48780488 0.21951220 0.04878049 0.75609756
```

Part 1: Understanding and implementing models

Now that we have our data, let's write some functions that will allow us to make predictions about people's judgments using different computational models.

We want our functions to take the events that were observed (the objects and activations), as well as any model parameters that might be relevant, and output the probability of each object being a blicket.

To illustrate, we'll start with a simple model that doesn't have any parameters.

Model: ΔP

Our first model is called ΔP; most of the code for this model is implemented for you in the block below.

Briefly, ΔP estimates the causal efficacy of a prospective cause by comparing the probability of the effect in the presence of the cause to the probability of the effect in the absence of the cause: $P(e = 1|c = 1) - P(e = 1|c = 0)$.

For details about this model, please look at page 339 of (Griffiths and Tenenbaum 2005) (link).

Let's see what the ΔP values are for our two experimental conditions, but we first need to complete the code below.

Question 1 (3 Marks): Complete the function `delta_P_values` in the code block below. You need only edit the line `delta_p <- 0.5` – no new lines of code are necessary.

```r
delta_P_values <- function(objs,activations,theta) {
  n_objects <- dim(objs)[2]
  n_events <- dim(objs)[1]
  delta_ps = rep(0,n_objects)
  for(i in 1:n_objects) {
    present <- objs[,i]
    active_with <- as.double(sum(activations[as.logical(present)]))
    p_with <- active_with/sum(present)
    active_without <- sum(activations)-active_with
    p_without <- active_without/(n_events-sum(present))

    # YOUR CODE BEGINS HERE
    delta_p <- p_with - p_without # Placeholder; replace it with your own code.
    # YOUR CODE ENDS HERE
    delta_ps[i] = delta_p
    if(is.nan(delta_p)) {
      delta_ps[i] = 0.5
    }
  }
  delta_ps
}
```

To be clear, ΔP values are not probabilities, but differences in probabilities that can be negative. Hence, we can't use these values as direct predictions of human judgments.

Question 2 (3 Marks): Briefly explain when ΔP might be positive and when it might be negative, and how this is likely to relate to blicket judgments given the way the experiment was set up.

Answer 2: ΔP is likely to be positive, when the probability of the effect is higher with the cause present than when it is not. ΔP is likely to be negative when the probability of the effect is higher when the cause is not present than when it is.

In relation to the experiment this means that ΔP will be positive when number of times the blicket detector is activated when the object is present more than the number of times the blicket detector is not activated when the object is present. This wil be vice versa for when ΔP is negative.

In order to arrive at judgment probability predictions, we need to transform the ΔP values appropriately. The function below includes a very naive version of this – how might you improve it, based on what you found in Question 2?

Question 3 (4 Marks): Define a function for converting ΔP values to predictions about human judgments.

Specifically, assuming that all people are the same, it should take a ΔP value for an object as an input, and output the probability that someone will judge that object to be a blicket. If it has any free parameters, pick sensible values for them in the body of the function and write a comment justifying your choice(s).

```r
  dps_to_predictions <- function(dp_vec) {
    library(scales)
    # YOUR CODE BEGINS HERE
    # abs(dp_vec) # Placeholder; replace it with your own code.
    dp_vec = rescale(dp_vec, to = c(0,1), from = c(-1,1))#scaled correlation delta p to probabil

    # YOUR CODE ENDS HERE
    dp_vec
  }
```

Question 4 (3 Marks): Explain why you made the changes you did.

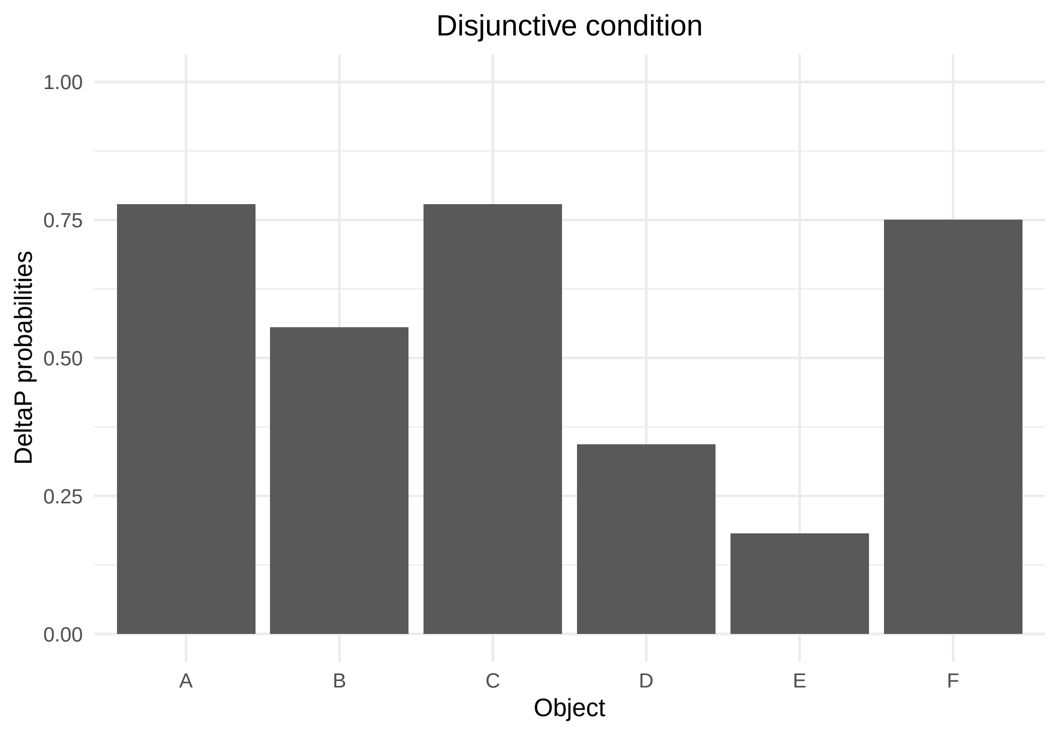
Answer 4: As ΔP will at most have a minimum value of -1 and a maximum value of 1, I have scaled the ΔP values to lie between 0 and 1 to hold the relationship between objects.

Let's look at our predictions, starting by defining helper functions to make it easier to see them:

```r
pp <- function(nums,cn='predictions') {
  names <- c("A","B","C","D","E","F")[1:length(nums)]
  ppdf <- data.frame(object_id=names)
  ppdf[cn] <- nums
  print(ppdf,digits=2)
}

plot_model_preds <- function(preds,model_name) {
  obj_names = c("A","B","C","D","E","F")[1:length(preds)]
  df <- data.frame(obj_names=obj_names,preds=preds)
  p<-ggplot(data=df, aes(x=obj_names, y=preds)) +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5)) +
    geom_bar(stat="identity") + ylim(0,1) +
      labs(y=paste(model_name,"probabilities",sep=" "),x="Object")
  p
}
```

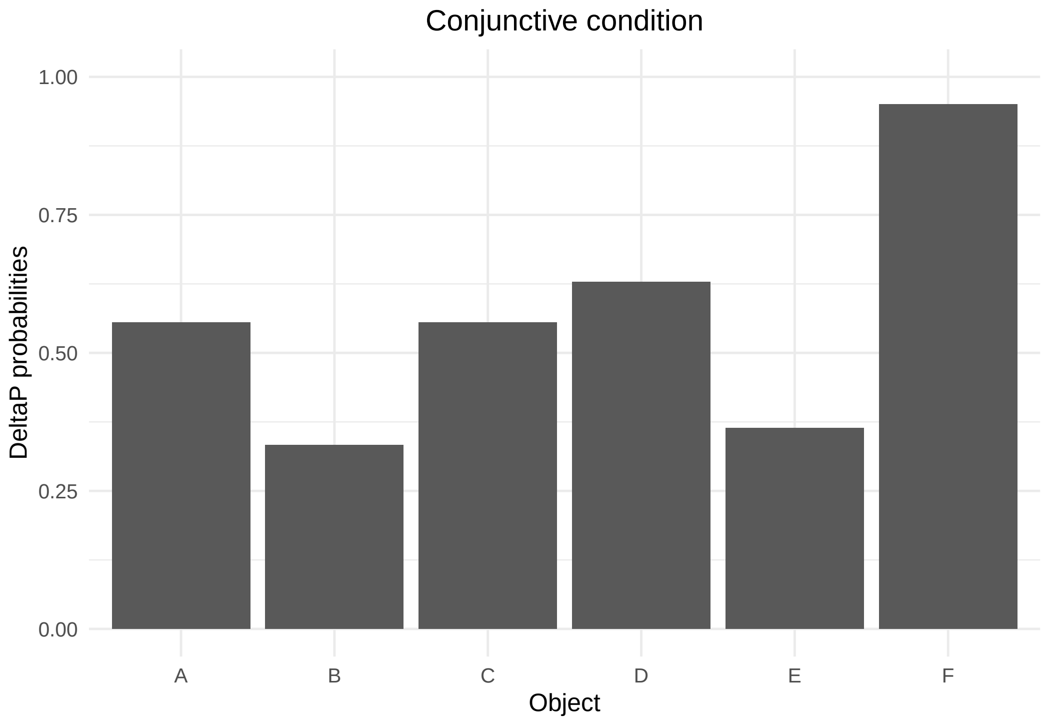
Let's see ΔP's predictions for objects in the DIS condition:



```
##   object_id deltaP
## 1         A   0.56
## 2         B   0.11
## 3         C   0.56
## 4         D  -0.31
## 5         E  -0.64
## 6         F   0.50
```

and in the CON condition:

```
##   object_id deltaP
## 1         A   0.11
## 2         B  -0.33
## 3         C   0.11
## 4         D   0.26
## 5         E  -0.27
## 6         F   0.90
```



In the next section, you will implement some alternative models, before comparing the different models to human judgments.

Model 2: Rescorla-Wagner

Originally published in 1972 as a model of associative learning, the Rescorla-Wagner model has also been offered as an account of human causal learning. We'll be following the notation for the RW model from (Danks 2003), and use the word `cue` interchangeably with `object` in this section, following the history of RW as a model of associative reasoning.

It has three kinds of parameters (the number of parameters depends on the number of cue objects being studied):

- α_i is the salience of the i^{th} cue object. We'll assume all alphas are the same (α).
- β_1 is the salience or "strength" of the effect occurring, while β_2 is the salience of the effect not occurring. Let $\beta_1 = \beta_2 = \beta$, for simplicity.
- λ is the maximum possible associative strength between any cause and the effect.

Question 5 (8 Marks): Implement a function that computes associations for the Rescorla-Wagner (RW) model, by editing the code block below. Let the first three elements of theta be α, β, and λ, in that order, and assume that all associations start at zero. Further, assume that the objects and activations were presented in the order they are saved as.

```r
rw_associations <- function(objs,activations,theta) {
  alpha <- theta[1]
  beta <- theta[2]
  lambda <- theta[3]
  n_events <- dim(objs)[1]
  n_objects <- dim(objs)[2]
  associations <- rep(0.0,n_objects)

  # YOUR CODE BEGINS HERE
    for(i in 1:n_events) {
      compound = 0

      for(j in 1:n_objects) #In the case where all associations are equal to 0
        if (objs[i,j] == 1){
          compound = compound + associations[j]
        }

      for(j in 1:n_objects)
        if (objs[i,j] == 1){
         associations[j] = associations[j] + alpha*beta*(activations[i] - compound)
        }


      }


  # YOUR CODE ENDS HERE

  associations
}
rw_associations(all_objects,con_activations,c(.5,.5,1.0))
```

```
## [1]  0.2500 -0.0625  0.1875  0.3750  0.0000  0.3750
```

Question 6 (4 Marks): Implement a function that generates predictions for the Rescorla-Wagner (RW) model, by editing the code block below. Your function should produce *judgment probabilities* so that its output can be plugged directly into your `human_judgment_scores` function. Justify briefly how your predictions can be interpreted as probabilities.

Answer 6: The associations for the RW model indicate how likely an object is to be a blicket or not a blicket between -1 and 1. Therefore I have rescaled the results to lie between 0 and 1 such that they become probabilities as to how likely an object is to be a blicket.

```r
  rw_preds <- function(associations) {
    ## YOUR CODE BEGINS HERE
    predictions <- rescale(associations, to = c(0,1), from = c(-1,1)) # Placeholder; replace it w
    ## YOUR CODE ENDS HERE
    predictions
  }
```

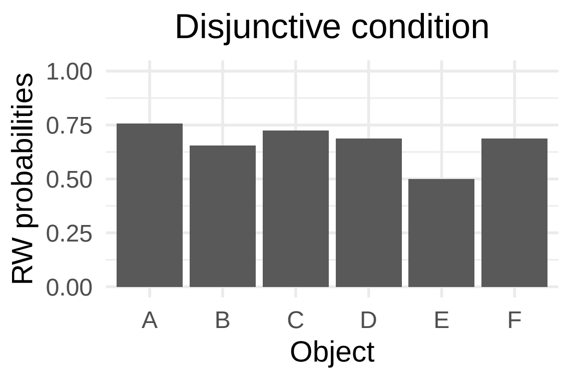
```
rw_preds(rw_associations(all_objects,dis_activations,c(.5,.5,1.0)))
```

```
## [1] 0.7578125 0.6542969 0.7246094 0.6875000 0.5000000 0.6875000
```

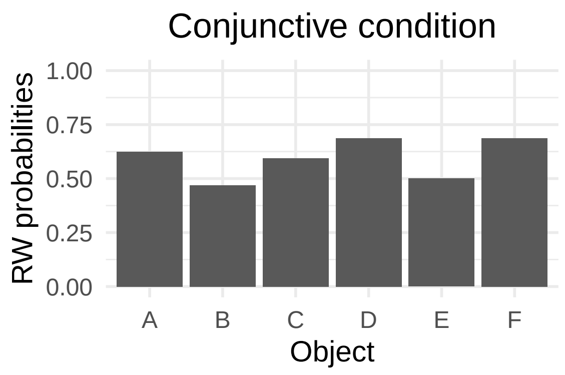
Visualizing predictions of the RW model

Let's visualize the RW model's predictions on the DIS data.

```
rw_preds_dis = rw_preds(rw_associations(all_objects,dis_activations,c(.5,.5,1.0)))
rw_preds_con = rw_preds(rw_associations(all_objects,con_activations,c(.5,.5,1.0)))
plot_model_preds(rw_preds_dis,"RW") + ggtitle("Disjunctive condition")
```



```
plot_model_preds(rw_preds_con,"RW") + ggtitle("Conjunctive condition")
```



Question 7 (4 Marks): Briefly (1-2 sentences) discuss a situation where the RW model makes predictions that differ systematically from human judgments. You may be able to find examples in the readings associated with this assignment.

Answer 7: The RW model fails to explain the backwards blocking phenomenon, for instance when two objects repeatedly turn the machine on, the model cannot create an association if one of the objects does not active the machine on its own. This differs as human judgement would learn that the object is indeed not a blicket.

Model 3: Griffiths and Tenenbaum's strength and structure

The last model we will look at is "causal support" (Griffiths and Tenenbaum 2005). First, make sure you understand the description of the model in the paper, and particularly the Noisy-OR parameterisation.

The form of the model used here has two continuous parameters:

- The causal power of the i^{th} cue object. We'll assume all objects have the same causal power (ws, in the paper, here referred to as `causalPower`).
- The prior probability of any object being a blicket.

We will use this model as an opportunity to use stan (Stan Development Team 2020). This model has already been implemented for you in `causal_support.stan` and the code to process the stan output is also provided below, however you will need to answer questions about the code.
Stan is a probabilistic programming language equipped with an efficient Markov chain Monte Carlo sampler (among other things) for performing statistical inference. If you are interested you can read more about Stan in the user guide. Stan lets us write down our generative model and has functionality for efficiently sampling from the posterior over model parameters. Here, we are interested in the posterior probabilities of each object (A-F) being blickets, based on the events and activations observed in each condition.

Sampling discrete latent variables is not supported in Stan (for an explanation and worked examples, see link1, link2), so we instead marginalise them out, and compute posteriors over the discrete latent variables afterwards.

In order to do this, we first compute all possible combinations of possible blickets, e.g., [0,0,0,0,0,0] for no blickets; [1,0,0,0,0,0] for `A` being the only blicket.

```
combination_lists <- rep(list(0:1), dim(all_objects)[2])
blicket_combinations <- data.matrix(expand.grid(combination_lists))
```

In the Stan code chunks below, we then pass the matrix of combinations as a parameter, can compute the posterior over whether objects are blickets with a few lines of R code based on the R output.

Question 8 (5 Marks) | Stretch: In line 51 of `causal_support.stan`, there is an additional background cause that is always present, no matter which objects are present and which objects are blickets:

`pAct = (1.0-(1.0-backgroundStrength)*pow(1.0-causalPower, liveBlickets[i]));`.

Why does the code include this additional background cause and what would happen if were to exclude this factor?

Answer 8: The code includes the additional background cause as as a constant in the case that if the prior probability is 0 such that calculated strength for an event is also 0, this prevents the case where the model will not learn from an event that it should be learning from.

Question 9 (2 Marks): Provide at least one reason why using Stan for the Causal Support model might be helpful over just writing the generative model in R like we did for the other models? Your answer can be brief (1-2 sentences).

Answer 9: Stan is more quicker to run, complex models like this can be computationally intensive and to create it in R can be resource heavy. R is built in C++ and outputs in R so it it will run much quicker.

Question 10 (6 Marks): Briefly state two conceptual differences between ΔP and causal support (1-3 sentences per difference), and one implications of each difference (1-2 sentences per).

Answer 10: The ΔP model is a measure of covariation and causal support is a measure an attempt to make ΔP assumptions explicit. Because of this the ΔP model is less consistent with human judgment than the causal support model. This is mainly due to the linear nature of the ΔP model.

The causal support model uses the Noisy-OR parameterization as to where the ΔP model does not. This allows for proper parameterization between causes and events allows for the causal support model to make better judgements than covariational model such as ΔP. This means that in general the causal support model is more consistent with human intuition when it comes to making judgements.

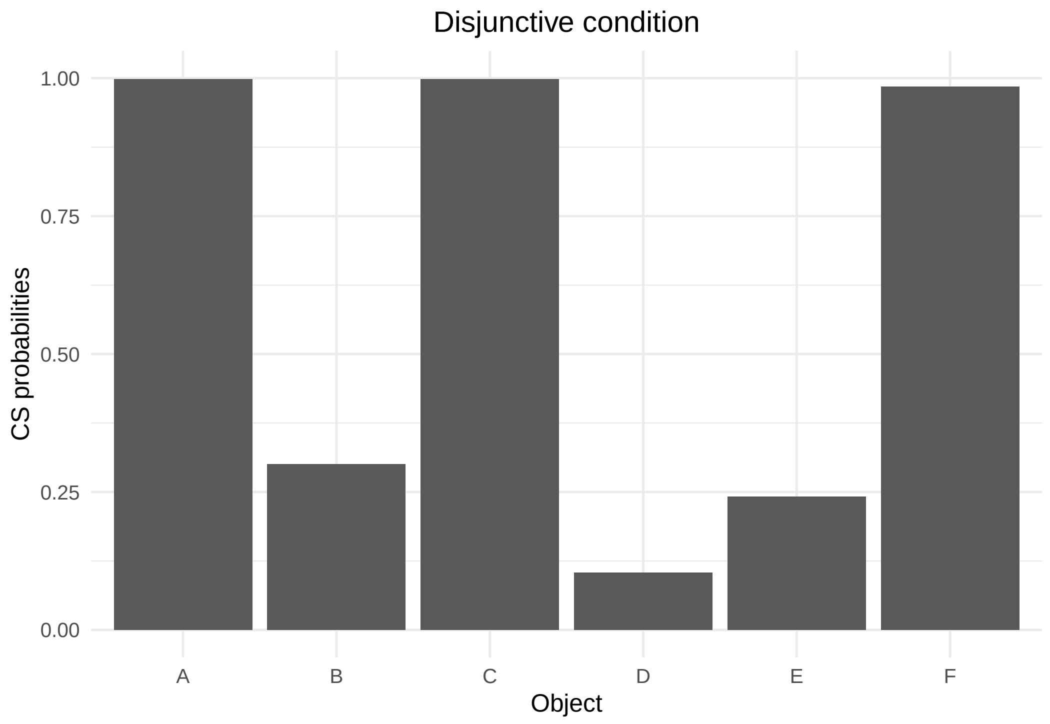
Posterior output from causal support model

Let's run our model.

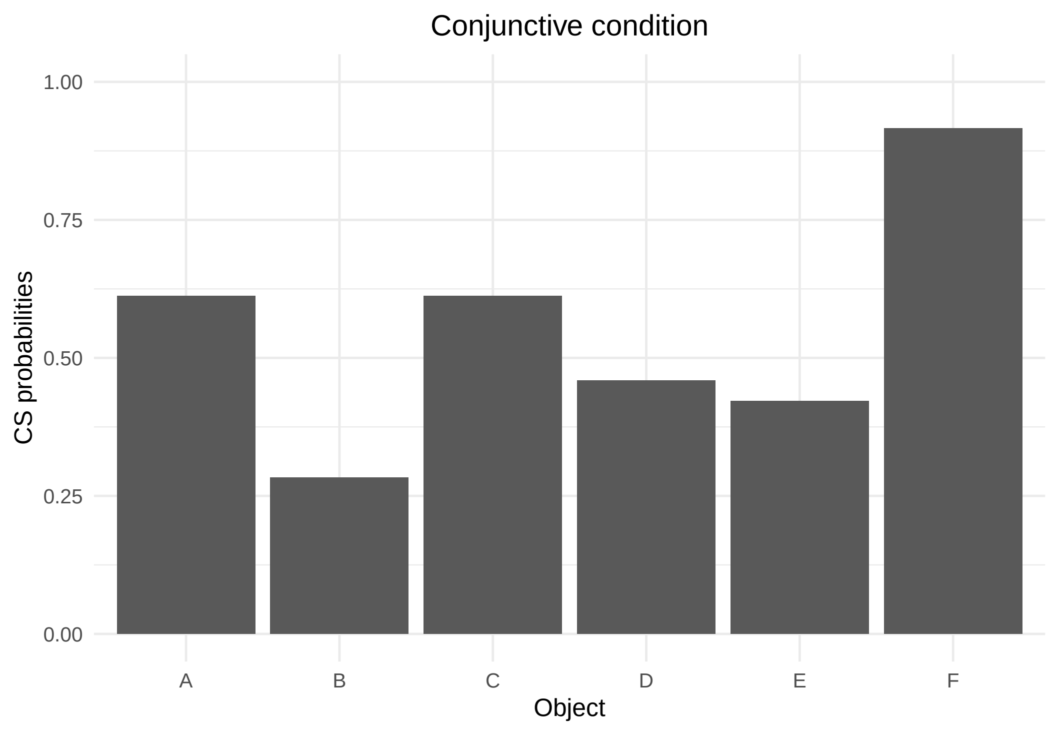
```
# This function may take a while to run.
cs_posterior <- function(objs, activations, blicket_combinations) {
  input_data <- list(nEvents=dim(objs)[1], nObjects=dim(objs)[2],
                     activations = activations, allObjects=objs,
                     nCombinations = 2**dim(objs)[2],
                     allCombinations = blicket_combinations)
  post_samples <- stan("causal_support.stan",
               refresh = 0, # suppresses intermediate output
               data = input_data,
               iter=3000)
  nps <- colMeans(extract(post_samples, permuted = TRUE)$np)
  cs <- c()
  for (b in 1:dim(objs)[2]){
    blicket_comb <- blicket_combinations[,b]
    np_select <- nps[as.logical(blicket_comb), drop=F]
    cs <- c(cs, sum(np_select)/(sum(nps)))
  }
cs
}
```

```
  cs_preds <- function(posterior) {
    posterior
  }
```

```
# Conjunctive case:
cs_preds_dis <- cs_preds(cs_posterior(all_objects,
                                      dis_activations, blicket_combinations))
plot_model_preds(cs_preds_dis,"CS") + ggtitle("Disjunctive condition")
```



```
# Disjunctive case:
cs_preds_con <- cs_preds(cs_posterior(all_objects,
                                      con_activations, blicket_combinations))
plot_model_preds(cs_preds_con,"CS") + ggtitle("Conjunctive condition")
```



Question 11 (5 Marks): Notice that we are not converting model outputs to judgment probabilities. What data model are we implicitly using here? (Note that Where `data` refers to people's judgments here.) State at least one alternative data model to the one being used.

Answer 11: Instead of outputting judgment probabilities the model is outputting log probabilities, this is the log-likelihood model. An alternative data model to use would be the Z-Score model.

Part 2: Comparing models

Let's see how well our different models (ΔP, RW, and Causal Support) predict human judgments.

Comparing models using probabilities

The function below assumes that the columns in the data structure are judgments for objects A-F, in order, corresponding to the prediction vector.

```r
human_judgment_probs <- function(dataset,prediction_vector) {
  joint_probability <- 1.0
  for(i in 1:length(prediction_vector)) {
    pColumn <- prod(dbinom(dataset[,i],1,prediction_vector[i]))
    joint_probability <- joint_probability*pColumn
    print(pColumn)
  }
  joint_probability
}
```

Here are the probabilities for the ΔP model:

```
human_judgment_probs(human_dis_data,dps_to_predictions(dp_dis))
```

```
## [1] 2.837828e-07
## [1] 2.033909e-19
## [1] 1.260207e-11
## [1] 1.88803e-12
## [1] 2.127295e-11
## [1] 1.493231e-08

## [1] 4.362377e-67
```

```
human_judgment_probs(human_con_data,dps_to_predictions(dp_con))
```

```
## [1] 4.372138e-27
## [1] 4.543879e-16
## [1] 9.941091e-26
## [1] 6.956698e-32
## [1] 8.544505e-18
## [1] 3.965186e-28

## [1] 4.654878e-143
```

Question 12 (5 Marks) Is there anything undesirable or unexpected about these values and how they're being computed here? How might you edit the function `human_judgment_scores` below so that model scores are still monotone functions of the total probability, while fixing some computational issues you might encounter with `human_judgment_probs`? Briefly explain what problem(s) your solution solves.

Answer 12: The values calculated are lower than expected, indicating that the ΔP model does not perform well according to this metric. The method used to score the ΔP model combines the probability of every object being a blicket for the dataset used. There are better techniques for model comparison that would give us a score that would be a better metric on how well this data model performs.

```
require(Math)
```

```
## Loading required package: Math
## Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
## logical.return = TRUE, : there is no package called 'Math'
```

```
human_judgment_scores <- function(dataset,prediction_vector) {
  joint_score <- 0.0
  # YOUR CODE STARTS HERE
  human_judgement_pred <- rep(0.0, length(dataset))
  cross_entropy <- rep(0.0, length(dataset))

  for(i in 1:length(dataset)){ #calculating the probability judgements for human judgement data
    prediction <- sum(dataset[,i]) / nrow(dataset)
    human_judgement_pred[i] <- prediction
  }

  for(i in 1:length(human_judgement_pred)){ #calculating cross entropy for each object of each m
    cross_entropy[i] <- human_judgement_pred[i] * log2(prediction_vector[i])
  }

  joint_score = -sum(cross_entropy)
  joint_score
  # YOUR CODE ENDS HERE
}
```

```
human_judgment_scores(human_dis_data, dp_preds_dis)
```

```
## [1] 1.738014
```

```
human_judgment_scores(human_con_data, dp_preds_con)
```

```
## [1] 1.014705
```

Now that we have a way to score our different model predictions, we can make our first model comparison. Above, we always looked at the CON and DIS conditions separately. We're now going to combine our predictions after we pass them through the `human_judgment_scores` function.

Question 13 (4 Marks) In the code chunk below, combine the scores from both conditions appropriately, such that you get an aggregate scalar score for both the DIS and CON conditions together.

```
combine_scores <- function(data, predictions) {
  scores <- mapply(human_judgment_scores, data, predictions)
  # YOUR CODE STARTS HERE
  mean(scores) # Placeholder; replace it with your own code.
  # YOUR CODE ENDS HERE
}
```

```
(delta_p_score <- combine_scores(list(human_dis_data,
                                 human_con_data),
                                 list(dp_preds_dis,
                                 dp_preds_con)))
```

```
## [1] 1.376359
```

```
(rw_score <- combine_scores(list(human_dis_data,
                            human_con_data),
                            list(rw_preds_dis,
                            rw_preds_con)))
```

```
## [1] 1.400459
```

```
(cs_score <- combine_scores(list(human_dis_data,
                            human_con_data),
                            list(cs_preds_dis,
                            cs_preds_con)))
```

```
## [1] 0.9881694
```

Question 14 (5 Marks): Suppose a colleague of yours argues that the Stan code above involved fitting parameters to data. What would you reply to your colleague? State whether or not the Stan code above fitted any parameters for the Causal Support model and make sure to justify your answer.

Answer 14: In response to a colleague, I would state that the model does not fit parameters, all parameters used in the causal_support.stan file are constants. The stan code above does not fit any parameters all the parameters used are predefined constants. For the model used there are 3 main parameters, backgroundStrength, causalPower and pBlicket; all of which do not change once the model has been called.

Question 15 (5 Marks): Report the rank-ordering of models from the one with the best predictions to the one with the worst predictive performance. Briefly justify how you compared the different models.

Answer 15: The models performance from best to worse is, Causal Support, Delta P and then Rescorla Wagner. I used cross entropy to compare the probability models, this allowed me to compare the models to the real human judgements. The higher the score the greater the uncertainty of the models predictions, This means that the lower the score the better the predictive power of the model.

Part 3: Fitting and interpreting parameters

In this section, we're going to illustrate how we may optimise model parameters on data we have observed. We already implemented all of our models and we have a way to score our models with the `human_judgment_scores` function. Now we need to decide which parameters we are going to fit. We are going to fit our parameters on both conditions simultaneously, and we are going to use the Nelder-Mead Simplex optimiser.

Question 16A (5 Marks): Complete the code chunk below to optimise parameters of the RW model. Make sure you understand how the `optim` function works. You need to optimise at least one parameter, but make sure to justify your choices for parameters to optimise and the fixed values for those you do not optimise. Report the optimal parameters you found as well as as your overall score when you use the optimised parameter, and note anything you might do differently or additionally if you were serious about trying to find a global optimum.

```
rw_eval <- function(theta) {
  # YOUR CODE STARTS HERE
  data <- list(human_dis_data, human_con_data)
  predictions <- list(
    rw_preds(rw_associations(all_objects,dis_activations,c( theta[1], theta[2], 1 ))),
    rw_preds(rw_associations(all_objects,con_activations,c( theta[1], theta[2], 1 )))
    ) # Placeholder; replace/edit with your own code.
  combine_scores(data,predictions)
}

(res <- optim(c(0.5, 0.5), rw_eval, method = "Nelder-Mead")) # Placeholder; replace/edit with yo
```

```
## $par
## [1] 0.7370908 0.7056503
##
## $value
## [1] 1.113281
##
## $counts
## function gradient
##       43       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
# YOUR CODE ENDS HERE
opt_params <- res$par
```

Answer 16A: For this question I decided to optimise the alpha and the beta parameters for the RW model. I chose these two parameters as they are weights for the salience of the object and the event occurring. I did not choose the lambda parameter as it is dependent on the outcome of the event and changes through iterations. The optimal parameters found for alpha and beta using the optim parameters are 0.7370908 and 0.7056503 respectively. The overall score for the model is 1.113281. I would have used a different optimisation method other than "Nelder - Mead" as there are better alternatives to use that would suit this problem.

Question 16B (3 Marks): Report the new rank ordering of the three models, now with fitted parameters for the RW model. Has the ordering changed, or is it still the same as above without fitting any of the RW model parameters?

```
# YOUR CODE STARTS HERE
rw_preds_dis_fit = rw_preds(rw_associations(all_objects,dis_activations,c(opt_params[1],opt_param
rw_preds_con_fit = rw_preds(rw_associations(all_objects,con_activations,c(opt_params[1],opt_param
# YOUR CODE ENDS HERE

(delta_p_score <- combine_scores(list(human_dis_data,
                                 human_con_data),
                                 list(dp_preds_dis,
                                 dp_preds_con)))
```

```
## [1] 1.376359
```

```
(rw_score_fit <- combine_scores(list(human_dis_data,
                              human_con_data),
                              list(rw_preds_dis_fit,
                              rw_preds_con_fit)))
```

```
## [1] 1.113281
```

```
(cs_score <- combine_scores(list(human_dis_data,
                              human_con_data),
                              list(cs_preds_dis,
                              cs_preds_con)))
```

```
## [1] 0.9881694
```

Answer 16B: The ranking for the 3 models has now changed. The ranking is now as follows from best to worst, Causal Support, Rescorla Wagner and then Delta P.

We now fitted our parameters on all data simultaneously. This might lead to problems, as we may just overfit to our data and have no way of knowing whether our RW model with fitted parameters would generalise well to unseen data based on the fitted scores alone. One option is to penalise our fitted scores. In that case, we could use an information criterion (like BIC) to penalise our model scores.

Question 17 (4 Marks): Explain how you would compute the complexity penalty for your fitted RW model. You can explain it in words, maths or code (R or pseudocode), as long as you make sure that your explanation would allow a colleague to implement and apply a solution in code. If you declare any variables, make sure to define them.

Another option for assessing generalisation performance would be to compute scores on a held-out test set. Here, we need to decide how we partition the data into training and test sets, such that can make claims about our model's ability to generalise to unseen data.

Answer 17: To calculate a complexity penalty for the fitted RW model I would use BIC to penalise more complex models, for both BIC and log likelihood are better therefore adding scores together will penalise more complex models.

```r
BIC <- function(dataset, predictions, nParameters){
  N = nrow(dataset) #number of observations
  k = nParameters #number of parameters
  LL <- rep(0.0, length(predictions)) #log likelihoods
  vPredictions = unlist(predictions) #vector of predictions

  for(i in 1:length(vPredictions)){
    LL[i] <- vPredictions[i]^2
  }
  LL = prod(LL)

  bic = -2*LL + log(N) * k


  bic
}

combine_scores_complex <- function(data, predictions, nParameters){
  score <- combine_scores(data, predictions)

  BIC = rep(0.0, length(predictions))
  for (i in 1:length(predictions)){
    BIC[i] = BIC(data[[i]], predictions[[i]], nParameters)
  }

  BIC = mean(BIC)
  penalised_score = score + BIC

  penalised_score
}

(delta_p_score <- combine_scores_complex(list(human_dis_data,
                                  human_con_data),
                                  list(dp_preds_dis,
                                  dp_preds_con), 3))
```

```
## [1] 13.99669
```

```r
(rw_score_fit <- combine_scores_complex(list(human_dis_data,
                              human_con_data),
                              list(rw_preds_dis_fit,
                              rw_preds_con_fit), 3))
```

```
## [1] 13.71534
```

```r
(cs_score <- combine_scores_complex(list(human_dis_data,
                              human_con_data),
                              list(cs_preds_dis,
                              cs_preds_con), 2))
```

```
## [1] 9.401811
```

```r
delta_p_score
```

```
## [1] 13.99669
```

```r
rw_score_fit
```

```
## [1] 13.71534
```

```r
cs_score
```

```
## [1] 9.401811
```

Question 18 (4 Marks) How would you partition the data into training and test sets, i.e., what should we generalise to: Unseen participants; unseen objects; unseen experimental conditions (or something else)? Describe your approach and make sure to justify your answer and state possible limitations.

Answer 18: For this experiment it makes the most sense to test using unseen participants. The test data used to test how similar the models guesses where to the participants guesses. A limitation of this is that human judgement is very unpredictable, meaning that small datasets can wield very different results each time they are used.

Part 4: Extensions

Question 19 (10 Marks) | Stretch Can you think of any extensions or new approaches for modelling human causal reasoning that go beyond the models we have considered in this course? Describe your proposed approach, briefly explain how your proposal would apply to the blicket experiment and what benefits it would have. It is fine to take inspiration from the literature as long as you reference it appropriately. However, please don't base your proposal on any of the course (co-)organisers' publications. *Do not write more than a page for this question.*

Answer 19: An extension of this experiment would be to use the models to make judgements on unseen objects based on events. I would use binary probabilistic models that and train them using the training data that we have. I would then run the models on unseen objects, events and activations and have then judge which object was the blicket. From this I would compare it to a large dataset of human judgement such that we have a good estimation of the average human judgement.

Other models I would consider using alongside the ones used previously are logit, MLE and naive Bayes. I would then score these using the methods I have including using the complexity penalty. The benefit we would have is we would be able to directly compare how close the models are to real human judgement. By using unseen objects we can then see how well the models perfrom at causal reasoning in comparison to humans.

Question 20 (8 Marks) | Stretch: We have focused on comparing a fixed set of well-known models. What are some limitations of this approach, or ways it could lead an incautious scientist or audience to incorrect conclusions? For full credit, your answer will go beyond the scope of what has been discussed in lecture and reveal insight and/or deeper familiarity with the scientific literature.

Answer 20: The limitation of using well known models is that readers assume that the models used where implemented correctly due to the volume of resources available. It can also lead to an overconfidence in the models used by the researchers as they may assume that the models work correctly and they may not question the results of the experiment.

Danks, David. 2003. "Equilibria of the Rescorla–Wagner Model." *Journal of Mathematical Psychology* 47 (2): 109–21.

Gopnik, A., C. Glymour, D. Sobel, L. Schulz, T. Kushnir, and David Danks. 2004. "A Theory of Causal Learning in Children: Causal Maps and Bayes Nets." *Psychological Review* 111: 1–31.

Griffiths, T. L., and J. B. Tenenbaum. 2005. "Structure and Strength in Causal Induction." *Cognitive Psychology* 51: 354–84.

Stan Development Team. 2020. "RStan: The R Interface to Stan." http://mc-stan.org/.