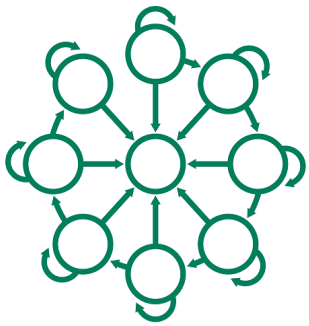


DECISION MODELLING FOR HEALTH ECONOMIC EVALUATION

Expected Value of Clinical & Diagnostic Information

Foundation Course Module 4

Jack Williams & Nichola Naylor

2022 Course

Overview

The aim of this exercise is to reproduce the calculations that underlie the slides presented in the didactic session, including trading sensitivity and specificity in order to choose an optimal point on the ROC curve.

We will use various R functions within the exercises, including:

- Data frames and matrices
- Apply function
- Writing our own functions and passing data to them as inputs
- Plots (base R, and ggplot for those interested)

We will use base R plots throughout the exercise, as these are quick and easy to produce, although they tend to be relatively simplistic. We will provide some more tailored graphs at the end of this exercise using additional packages.

Step by step guide

The exercise will be split into four parts:

1) Setting up the parameters of the model
2) Expected value of perfect diagnostic information (EVPDI)
3) Test accuracy for imperfect tests
4) Expected value of clinical information (EVCI)

1. Setting up the model parameters and estimating NMB

We start by defining the consequences / payoffs for the simple decision tree model presented in the videos. Please use the "F4.3.2_Diagnostics_Template.R" file.

i) First, define the prevalence (0.3, i.e. 30%) and a willingness to pay threshold (£30,000).

ii) The expected outcomes for sick and healthy individuals who are treated and not treated are provided, and then combined in a data frame that gives the payoffs and consequences associated with the four possible pathways. These costs and outcomes are provided in the parameter values data frame, which is shown below.

```
##                outcome.names expected.cost expected.qaly nmb
## 1        Sick person treated          6000          0.80  NA
## 2    Sick person not treated          5000          0.50  NA
## 3     Healthy person treated          3000          0.95  NA
## 4 Healthy person not treated          1000          1.00  NA
```

iii) There is a blank column for the net monetary benefit (NMB) in the `parameter.values` data,frame. Calculate the expected NMB payoff based on the costs and QALYs associated with each outcome, based on the prevalence assigned earlier. The NMB for Sick person treated should be equal to **£18000** . *(Hint: The expected NMB will be important throughout the exercise, so make sure to calculate this correctly).*

Now that you have calculated the NMB for each outcome, this can be used to estimate the NMB for a strategy of 'treat all' or 'treat none'.

iv) Based on the parameters table, create a data.frame comparing the costs, QALYs and NMB associated with the two strategies. Note that these NMB calculations are specific to the prevalence of the condition. You can define this data frame as the `expected.values` table, and the costs have been calculated for you. Calculate the QALYs using the same calculation as the costs, and then calculate the NMB (`expected.values$nmb`), which has also been left blank.

v) The next step will involve creating a function to estimate the NMB for the two strategies (treat all versus treat none) in the absence of a test. The function will be called `est.nmb`. Creating a function allows us to perform the calculation many times easily, by wrapping all of the code up into a set function, and passing arguments (e.g. inputs) to this function.

The function will take the parameter values (defined at the start of the exercise) and the lambda value (i.e. the willingness to pay threshold) by default. The function also requires the prevalence of the condition to evaluate the NMB.

The calculations within the function are provided for you, but make sure that you understand how the costs and outcomes are being calculated for those who are healthy and sick (and treated or not treated). The calculations are the same as those defined in point (iv) above, except that this time they are within the function.

This function is labelled `est.nmb()` and the main argument that we need to provide is the prevalence (named `prev` within the function). Try running the function by providing different prevalence values, and see how the estimated NMB changes for 'treating all' and 'treating none'. Is this what you expected?

vi) Now that you have run the function with different prevalence values, create a vector of prevalence values ('`prevalence.vector`'), from 0% to 100% (or 0 to 1) by 5% increments (you can use the `seq()` function to do this). Pass the vector of prevalence values to the `est.nmb()` function. The function will estimate the NMB associated with each prevalence for the vector provided. Save the NMB values (i.e. the output of the function), so that we can create a plot to help visualise this change.

vii) Now you can create a simple line plot in base R, showing the NMB for the two strategies (treat all vs. treat none). *Hint: use the `plot()` function, with additional `lines()` in order to plot both "Treat all" and "Treat none".*

2. Expected value of perfect diagnostic information

If a perfect test were to exist then all sick patients would receive treatment and all healthy patients would remain untreated. We will use a function to estimate NMB, similar to the function used to estimate NMB in the absence of a diagnostic test.

In this example, we will assume that all sick patients are treated, and that all healthy patients are not.

i) Based on `est.nmb()` have a go at coding the `est.nmb.perfect.test()` function, we've outlined the structure for you.

ii) Run the function, using the 30% prevalence. How does this NMB compare to the previous function, and why? *(Hint: You can run the the function using the 30% value two ways, either as shown below, to get the same output)*

```
est.nmb.perfect.test(prev = 0.3)
```

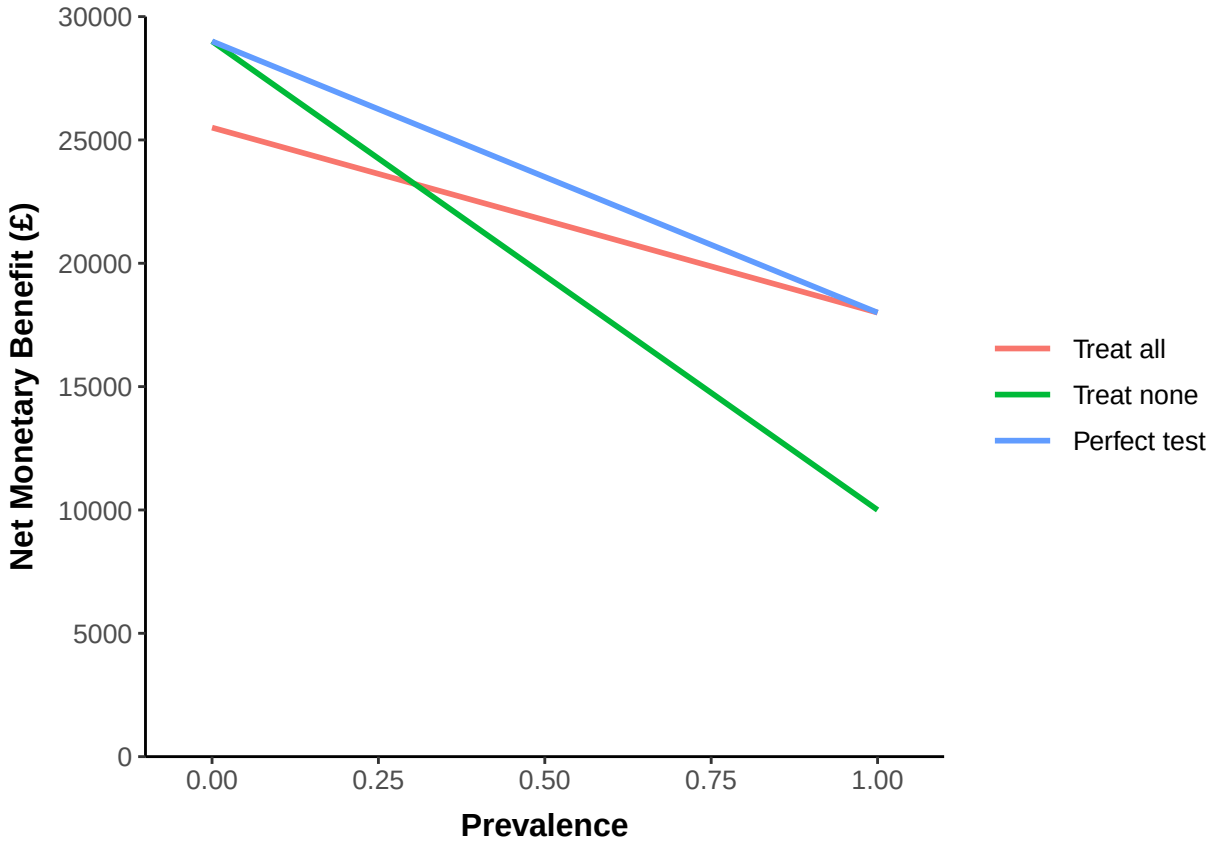
```
##   prevalence   nmb
## 1        0.3 25700
```

```
est.nmb.perfect.test(prev = prevalence)
```

```
##   prevalence   nmb
## 1        0.3 25700
```

iii) Now run the function using the vector of prevalence values that you created earlier (`prevalence.vector`), to get the NMB of a perfect test, across different prevalence values. Make sure to save this as a data.frame - you can save this as `perfect.test.nmb`.

iv) Next, compare the results generated in the `perfect.test.nmb` to the results saved earlier in the `no.test.nmb` data frame visually by filling in the plot function we've left blank. You should be able to see that the NMB for the perfect test is higher than the treat all or treat none strategies (except when the prevalence is 0 or 1). Add another line to the line plot created previously (again using `plot()` and `lines()` for now), '`prevalence.vector`'. Your plot should look like this (don't worry if the formatting is slightly different):



The final step will involve estimating the Expected Value of Perfect Diagnostic Information (EVPDI) - which is simply the difference between the expected value of a perfect test and the expected value of the optimal course of action ('Treat All' or 'Treat None') in the absence of a test.

vi) Before being able to do this, you will have to estimate the maximum NMB for the two strategies without a test (treat all or treat none), and you will need to do this across the prevalence values in the 'no.test.nmb' table. The code has been provided, which uses an apply function to estimate the maximum value in those two columns (i.e. the NMB for two strategies), for each row of the data frame. The code is provided below - check to make sure you understand how the apply function does this.

```
no.test.nmb$nmb.max <- apply(no.test.nmb[,2:3], 1, max)
```

vii) Finally, You can do calculate the EVPDI by simply subtracting the max NMB without a test from the NMB with a perfect test, and save this as 'evpdi'. When the EVPDI results are added to a data frame with the prevalence, this is what the results should look like (with just the first 6 rows showing below):

```
head(evpdi.table)
```

```
##   prevalence evpdi
## 1       0.00     0
## 2       0.05   400
## 3       0.10   800
## 4       0.15  1200
## 5       0.20  1600
## 6       0.25  2000
```

3. Test accuracy for imperfect tests

Diagnostic tests typically measure the value of a biomarker that is present in both healthy and sick individuals – for example blood glucose levels in diabetes. However, it is rare that such biomarkers perfectly distinguish those with the condition from those without.

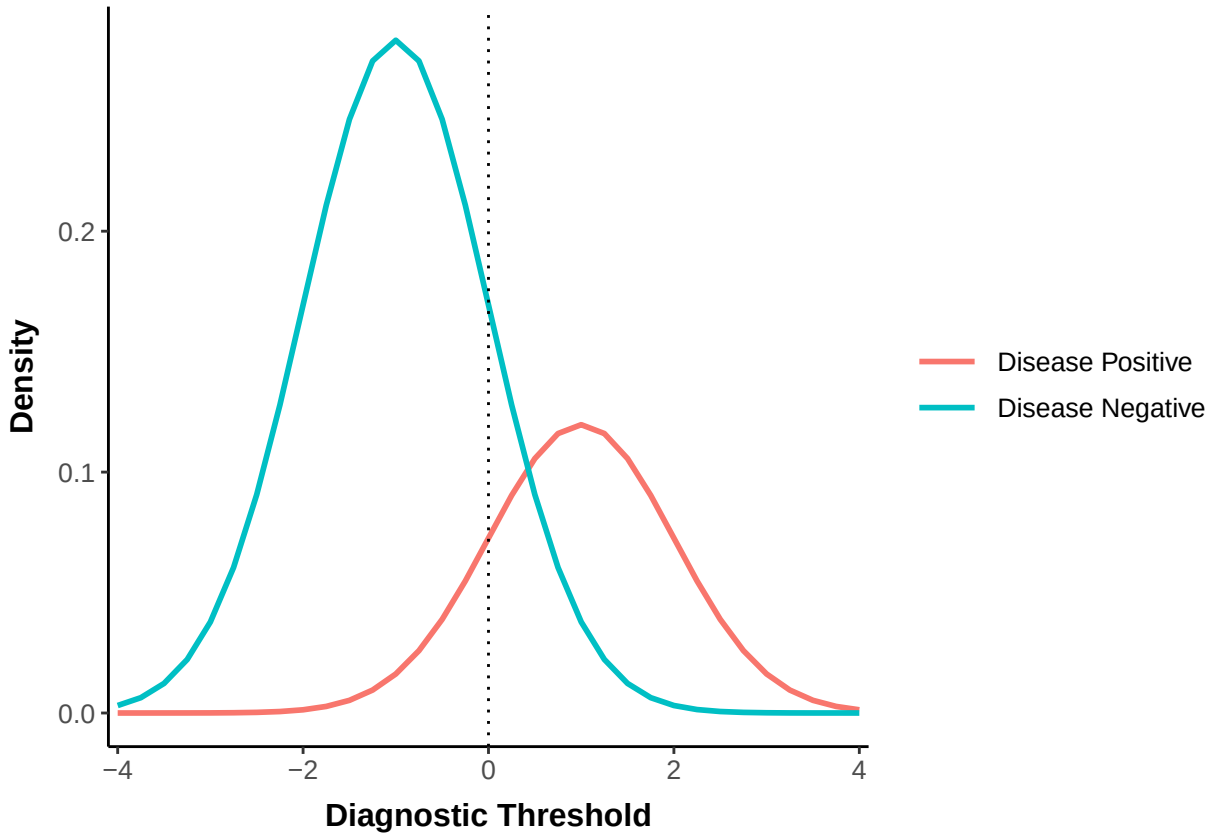
We can define the test characteristics, using a mean and standard deviation for the biomarker. These are provided in the `test.char` vector within the template file, and also shown in the table below. The hypothetical numbers given in the table below (such as a mean value of "biomarker indicator" in positive disease = 1) are (highly) stylised distribution parameters for a biomarker in the disease positive and disease negative populations. These parameters assume the biomarker is distributed normally in both populations given the mean and standard deviation parameters defined, and the prevalence of disease.

	Mean	SD
Marker in positive disease	1	1
Marker in negative disease	-1	1

i) First, create a vector of diagnostic threshold values from -4 to 4, in increments of 0.25. Use the `seq()` function to do this. This can be saved as `diagnostic.threshold`.

ii) Next, using the mean and standard deviations of the test characteristics for positive and negative disease (within the vector `test.char`), the disease positive and disease negative distributions can be calculated. Using the normal distribution (this is the `dnorm()` function, for the probability density of the normal distribution), which requires the diagnostic threshold values assigned in point i, and the mean and SD parameters of the distribution (type `?dnorm` in the console for more information) we have coded this for you. These can then be stored in the `biomarker.dist` data frame.

Plot the distributions on a chart and verify you get the same as those presented in the teaching session (you will need to set the prevalence to 30% if you have changed the original prevalence defined earlier on in this exercise). The distributions will look like this (again don't worry that the formatting is slightly different):

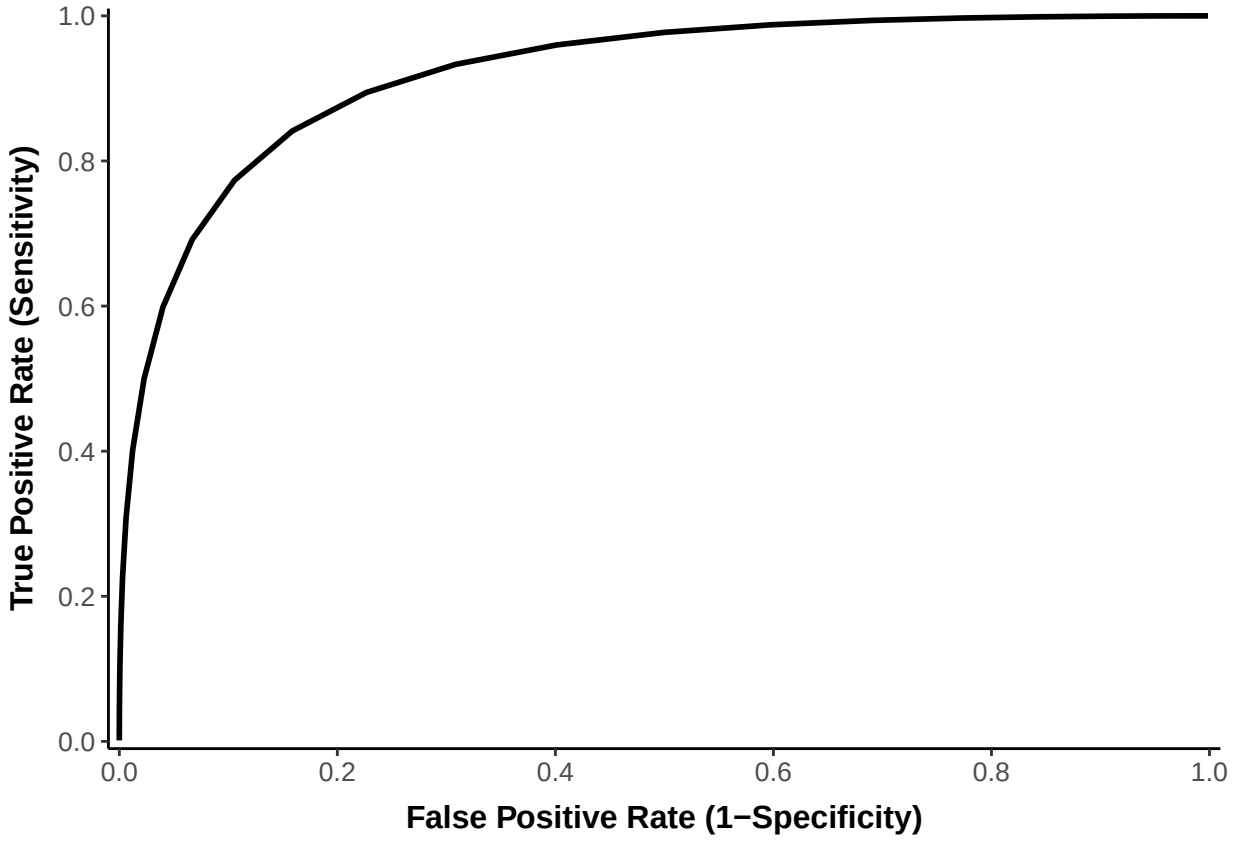


iii) Assuming a diagnostic threshold of 0 (as indicated by the dotted, vertical line in the above plot), ensure you can identify the True Positives, True Negatives, False Positives and False Negatives as per the video presentation. *(Hint: type ?pnorm in the console to see what parameters you need to enter)*

iv) Now we can estimate the TPR - True Positive Rate (i.e. sensitivity) and the FPR - False Positive Rate (1- specificity), using the cumulating normal distribution, which is the `pnorm()` function in R. The function requires you to add the same data for the normal distribution (the diagnostic thresholds, and the mean and SD of the test characteristics). Save these as 'TPR' and 'FPR'. *(Hint: make sure you understand why the cumulative normal distribution is used here)*

Note that when we code the TPR and FPR we use `1 - pnorm()` in the R code because by default the `pnorm()` function gives the probability of being to the left of the X value, whereas we want the probability of being to the right. Another option is to add `lower.tail = FALSE` as an argument into the `pnorm()` function so that it returns the probability of being to the right of the X value, which avoids the need to to use `1 - pnorm()`. For those interested, we have added both into the solutions script to show they both return the correct probabilities.

v) Now save the TPR and FPR in an data frame alongside the diagnostic threshold, and create a Receiver Operating Characteristic (ROC) curve (using the code already provided). The figure will look like this:



4. Expected Value of Clinical Information

Just as we did for a perfect test, we can also calculate the value of information provided by an imperfect test. However, this time we need to allow for the error rates (false positives and false negatives) that will result in some patients not getting optimal treatment.

i) Using the error rates specified earlier (True Positive Rate and False Positive Rate), calculate the pathway probabilities for true positives (`TP`), false positive (`FP`), false negatives (`FN`), and True negatives (`TN`). Note that these probabilities are based upon the prevalence of disease, and the equations are provided for you for some, fill those currently blank.

ii) Next, you can now use the TP, FP, FN, and TN to calculate the expected NMB of the imperfect test at each diagnostic threshold. This will use the same method as was used to calculate the NMB without a test and the NMB for the perfect test. Calculating the NMB of the imperfect test will involve getting the NMB payoff parameters assigned at the start of the exercise (in `parameter.values` data frame). To remind you, these are shown below:

```
##                outcome.names expected.cost expected.qaly   nmb
## 1         Sick person treated          6000          0.80 18000
## 2     Sick person not treated          5000          0.50 10000
## 3      Healthy person treated          3000          0.95 25500
## 4 Healthy person not treated          1000          1.00 29000
```

iii) First, we've created a blank vector to store the total NMB results across each diagnostic threshold.

iv) Now, complete a loop to calculate the NMB associated with each TP, FP, FN, TN, and sum these to give the total NMB. This is the NMB of the imperfect test across the diagnostic thresholds. The loop has been started for you, but needs to be completed. *Note: There are other ways to estimate the NMB, instead of using a loop. One example is to use apply, and an example has been provided in the script.* Once the results have been generated, you can save this vector into the 'diagnostic.accuracy' data frame, by adding the vector as a new column.

The 'optimised' trade-off between sensitivity and specificity occurs when the NMB across the diagnostic threshold is maximised. The difference between this value and the optimal strategy in the absence of a test ('Treat all' or 'Treat none') is the EVCI (Expected Value of Clinical Information).

Since we want to calculate the EVCI for any given prevalence, a function is provided (`est.evci`) that will estimate the TPR and FPR, and estimate the NMB at each diagnostic threshold (using the diagnostic accuracy data). The first part of the function essentially repeats the calculations above (i.e. estimating TPR and FPR, and using this to calculate the overall NMB across true/false positives/negatives).

However, the function also calls an earlier function within in. This is the `est.nmb` which was used to estimate the NMB in the absence of a test.

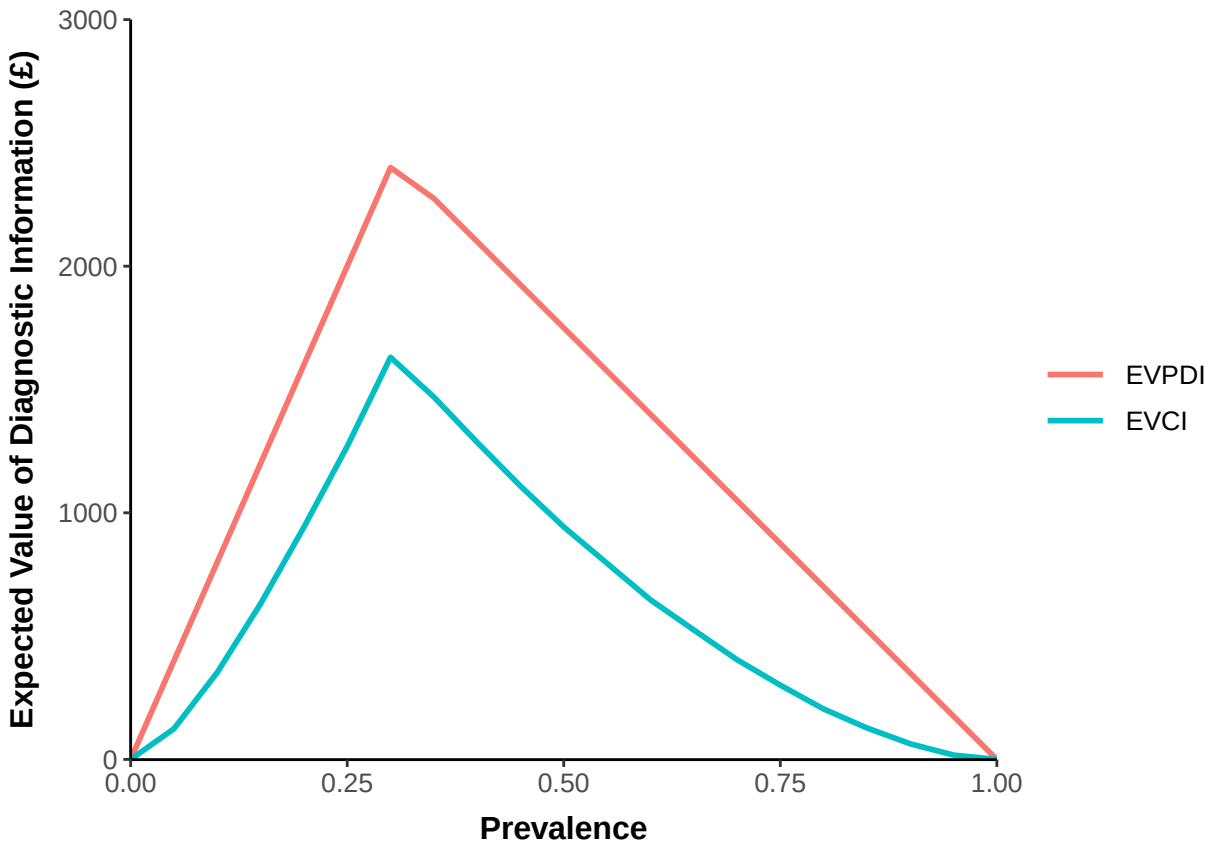
Remember, this function allows us to pass prevalence values (arguments) into the function, so that the function can estimate the EVCI at different prevalence values provided (or a vector of prevalence values).

v) First, remind yourself of the output of the `est.nmb()` function, and then look through the `est.evci` function to check that you understand the calculations provided in the function.

vi) Use the `est.evci` function and pass different prevalence values to it, e.g. 30% prevalence, to estimate the maximum NMB associated with 'no test' (the higher NMB of the treat all or treat none strategy), and the NMB associated with the imperfect test. The difference between these values gives the EVCI.

vii) After checking the EVCI results at different levels of prevalence, pass the 'prevalence.values' vector into the function, to estimate the EVCI across a range of prevalence values.

viii) Finally, you can produce a plot (with the code provided) to compare the Expected Value of Clinical Information (EVCI) with the Expected Value of Perfect Diagnostic Information (EVPDI) that was estimated with the perfect test. Remember that both of these are the value compared to no test. The plot will look like this:



Code for running ggplots

We've added some additional code at the bottom of the template and solution files that use the `reshape` and `ggplot2` packages to show you other potential ways to plot the EVDPI, EVCI, Diagnostic thresholds and ROC. If you have not installed these packages before, you will need to install the following packages for ggplot and to help reshape the data:

```
install.packages("ggplot2")
install.packages("reshape2")
```

'ggplot' is a graphics package that allows you to make very good graphs and plots, but requires the data to be structured in a specific way (long format), and tends to require more detailed code (which is why it allows for highly detailed plots). For more information on ggplot2, see the cheat sheet available online: https://www.rstudio.com/resources/cheatsheets/ - see "Data Visualization Cheatsheet"