Specifying regression

A typical notation

This article offers some suggestions on how to specify regression models and rules for making inferences about models from data. It provides an alternative to a long established style that I find particularly confusing and bizarre, which involves presenting a model something like this:

$$y_i = \alpha + \boldsymbol{\beta}\boldsymbol{X}_i + \varepsilon_i, \quad \text{where } \varepsilon_i \sim N(\mu, \sigma^2)$$

What is wrong with this notation?

This notation tends to produce confusion in the following ways:

Confusion between models and model families: The 'models' described are usually formulae with parameters represented by letters (often ancient Greek letters), not by numbers. In the example above, the parameters are α, $\boldsymbol{\beta}$, μ, and σ. A specific model would be one where these parameters have specific values. When Greek letters are written instead the formula usually represents a family of models that have the same formula but with different values for parameters. The phrase 'fit the model' actually means 'find the model within the model family that looks like the best choice, given these data.'

Confusion over what the index ranges over: In this style a general rule is presented *almost* as if it were just one example. The fact that this is a general rule is conveyed only by the suffixes and, concerning those, some important information is often unstated. What does i represent? Is this equation true for all the data we already have (and will use to train the model), or is it true for all imaginable cases, or some other set? A statement that starts with '$\forall\, i : \mathbb{N} \mid i \in \cdots$' (pronounced as 'For all i, where i is a natural number and a member of the set of …') might provide the missing information but is often missing.

Confusion over what 'distributed' means in this context: In this notation the idea is that all the terms except N represent numbers or matrices, not distributions or anything more complex. The last term, ε_i, is special because it represents the numbers that make each equation balance and are also in some way 'distributed' Normally.

Despite the claim that $\varepsilon_i \sim N(\mu, \sigma^2)$, which is pronounced '$\varepsilon_i$ values are distributed Normally with parameters μ and σ', the reality is that ε_i represents a vector of numbers, one for each equation. A vector of numbers like this is not a distribution, though they may look like numbers that might have been selected by generating random numbers according to the Normal distribution. A set of numbers that could have been drawn like this is not the same, logically, as a probability density function and some vectors might be more typical of draws from a Normal distribution than others, yet still plausible, or at least possible.

Confusion between the statistical model to fit and the model of reality assumed: The example formula above is not the model, or even the model family, that is typically 'fit' to the data. The model that is 'fit' excludes the ε_i term. (Sometimes it is not even clear whether the model will predict a specific value for y or the probability that any given value for y is seen, given the value of X.)

This reflects an often-used pattern of development in statistics: start by making a bundle of assumptions about the system to be modelled, then make deductions given those assumptions. With statistical hypothesis tests you make all those assumptions, plus a 'null hypothesis' assumption, and on the basis of all those you calculate the probability of the data you have. If it seems very unlikely that the data could be the result of a system that meets all your assumptions you can conclude that the null hypothesis very probably is not correct *provided all the other assumptions are met*. Often the other assumptions are as doubtful as the null hypothesis and so the test is not really applicable.

Making extra assumptions is usually unavoidable with regression models that only make point predictions rather than probabilistic predictions. Strictly speaking, if a model cannot predict all the data in your data set then that model is wrong. It cannot be correct. The probability of its being true is zero. But with point prediction regression models it is traditional practice to assume that there are other factors not captured in the model that drive the outcomes and just try to minimise the gaps between the predictions and reality. That often means minimising the square of the differences between predictions and data. Justifying the choice of fitting rule usually involves making assumptions about the data that, if true, would make the fitting rule optimal in some way.

A Bayesian approach using a probabilistic regression model makes such additional assumptions unnecessary. All relevant assumptions are in the model itself.

In the notation to which I am objecting, the significance of the full formula (such as the one given to start above) is not usually clear but is either a combination of the model family plus an additional assumption about the data, or is a statement of the assumed true process that is generating the data. It is confusing to mix information about the statistical model to be applied and the assumed process that generates the model. This confusion is all the greater when the process is described with mathematical modelling ideas, not in terms of the real entities involved, such as traders on a market, or crops in fields, or birds on an island.

In short, this style confuses models with reality. It also encourages confusion over what the statistician already knows or assumes about the process being studied and what has to be inferred from the data.

Missing information: The rule for 'fitting' the model to data is often not specified and often is crucial to the results achieved. The calculations to fulfil the fitting rule are also frequently unstated even though sometimes they are very complicated and, again, crucial to the results achieved.

An alternative style

A clear and correct way to specify regression should clearly separate information about the various objects used in regression modelling. It should avoid confusing assumptions about the situation with features of the mathematical model, or with the method of fitting. It should do this without any unspecified or clashing mathematical types.

The following need to be clearly distinguished and specified:

- The **data** used, which can be in various different structures, and will usually include a training set, used to set up the model, and a test set or prediction set, used to elicit new predictions from the model.
- Simple **data transformations** to use on those data, such as differencing.
- **Model families** and **model instances**, being formulae and formulae with specific numbers for parameters respectively.
- **Model learning rules**, such as Maximum Likelihood Estimation and Bayesian updating.
- **Model learning rule proofs**, to show that the model learning rule, if adhered to, provides good results. This might include additional assumptions about the real system, such as the distribution of differences between the model's point predictions and reality.
- **Model learning processes**, such as formulae, iterative processes, and statistical hypothesis tests (implausibility inferences). These will be chosen to try to meet the requirements of the model learning rule.
- **Model learning process proofs**, to establish that model learning processes work, or that they work to a particular extent.
- **Model performance measures**, used to evaluate regression models in practical applications.

Where the Bayesian model learning rule is used the result is not one selected model, but a probability distribution linking every model considered to its probability of being the best model. So, in order to use this information in a decision, or to make a prediction, the list of things to be specified may also need to include:

- A **prediction combination rule**, such as Bayesian Model Averaging, to combine the predictions of each possible model into one.
- A **decision making rule** that somehow combines the information given by calculating the decision using each possible model.

Guidelines to implement this style and so improve the clarity of regression specifications are as follows:

- Use the terms in bold (above), consistently so that readers know what they are being given at each point.
- Keep the elements of the specification separate so that there is no confusion as to the status of formulae presented.
- Miss no element out, by accident. If an element is too complex, lengthy, or outside the scope of a document then reference to another source or acknowledge the gap.

Here is an overview showing some of the well-known techniques classified.

	Point predictions	**Probabilistic predictions**
Type of prediction	Model's prediction is just a single value – a 'best estimate'.	Model's prediction is a probability distribution or probability density distribution that can give the probability or density of particular values.
Model families	Multiple linear regression, common time series models (AR, MA, ARIMA, ARCH, GARCH, EWMA, Holt Winters)	General Linear Model, linear function plus noise, Monte Carlo simulation of a point forecasting model
Model learning rules	OLS	MLE, Bayesian
Model learning processes	OLS formula, iterative search	Iterative search, Bayes formula, MCMC, Bayes factors
Model learning process proofs	OLS formula proof, proofs about asymptotic behaviour, experiments	Bayes formula proof, proofs about asymptotic behaviour, experiments
Model performance measures	RMSE, MAPE	Brier Score, IGN, Information Gain

And here are some well-known types of data and transformations of data.

Data structures	Set of cases (Xs with Y), time sequence, multiple time sequences, panel data
Data transformations	Differencing, logs, percentage differences, log of percentage differences, arithmetic and geometric returns

The following section gives illustrative specifications for some common situations.

Examples

Example 1: A simple linear model

The first example begins with specification of the **data** involved. Imagine that we have one variable that is going to be used to predict another. For example, perhaps we are trying to predict the pressure inside a tyre from the temperature inside it. In this example both variables are numbers.

We have some cases already where we know the value of both variables, so that enables us to learn about the relationship between them. These data form our training set. The type of a pair of values, one for each variable, is a pair of real numbers. To make it easier to distinguish visually between the predictor and predicted variables, we define two types, both of which are just alternative names for a real number[1].

$X \;==\; \mathbb{R}$

$Y \;==\; \mathbb{R}$

X now represents the type of the predictor variable while Y represents the type of the predicted variable.

However, our training set is not necessarily a set of unique pairs of values because some pairs might be identical, but it is still important to distinguish between them. So, we represent the training set as a 'bag', which has the number of times each pair occurs, for each pair that actually does occur at least once. This formula gives the name of the training set, tr, and specifies its mathematical type.

$tr : (X \times Y) \rightarrow \mathbb{N}$

Having specified the data involved it is time to specify a model instance and the model family that creates it. A specific **model instance** in this type of regression takes as input a number, representing a possible value for the predictor variable, and it returns a number, representing the predicted variable. Here is its name and type.

$m : \; X \rightarrow Y$

This model, m, gives a best estimate for the predicted number rather than a more useful and informative probability distribution for possible values.

The **model family** is a function that creates this type of model. It has two parameters: the inputs to the model family function, α and β, both of which are real numbers. It returns as output a model instance that makes predictions. Here I have called the model family slm, which is short for simple linear model.

$slm : (\mathbb{R} \times \mathbb{R}) \rightarrow (X \rightarrow Y)$

The **model family** is specified like this:

$\forall\, \alpha, \beta : \mathbb{R}, x : X \cdot slm[\alpha, \beta][x] = \alpha + \beta \times x$

[1] The notation used here is the mathematical toolkit of Z (see Spivey, 1989). It is longer than the usual notation but not necessarily harder to understand. It leaves nothing unsaid, which is a great advantage when you want to know and understand everything.

In words, for all α and β, and all x, where x is of the type X (a real number), the model output for x as input is equal to $\alpha + \beta x$. Instances of this model family are models with specific values for α and β. For example,

$$\forall\ x : X \cdot slm[2,3][x] = 2 + 3 \times x$$

The rule by which the relationship between the two variables is learned and encoded into the model is conventionally one that finds values for α and β that minimise the value of the squares of the differences between the model's predictions and the actual data, summed across the training set. This is the OLS criterion, short for Ordinary Least Squares.

This **model learning rule** is specified as follows:

$$\alpha_{OLS}, \beta_{OLS} :\ \mathbb{R}$$

$$\neg\exists(\alpha', \beta') : \mathbb{R} \mid sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times \big(y - (\alpha' + \beta' x)\big)^2] \\ < sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times \big(y - (\alpha_{OLS} + \beta_{OLS} x)\big)^2]$$

This rule says that α_{OLS} and β_{OLS} are values for the parameters of the model such that there is no other pair of values with a lower sum of squares of differences between prediction and predicted values, summed across the training set. Without doing more mathematics, it seems that there could be more than one pair of values of the parameters that has this property.

The purpose of a **model learning rule proof** is to show that the model learning rule, if adhered to, provides good results. Quite a lot of work has been done with this model family and model learning rule and several properties of their performance have been proved. This material is too complicated and lengthy to include in this illustration.

The **model learning process** is simply to calculate the parameters using formulae, specified as follows:

$$\bar{x} = \frac{sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times x]}{sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)]]}$$

$$\bar{y} = \frac{sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times y]}{sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)]]}$$

$$\beta_{OLS} = \frac{sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times (x - \bar{x}) \times (y - \bar{y})]}{sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times (x - \bar{x})^2]}$$

$$\alpha_{OLS} = \bar{y} - \beta_{OLS} \times \bar{x}$$

The **model learning process proof**, in this case, is simply a proof that the formulae do indeed provide the minimum sum of squared differences. It involves partial differentiation to find the values that minimise the sum of squared differences. Here is a simple version where x represents just one number, not a vector of predictors as in the general model family. Let S be the sum of squared differences.

$$S = sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times \big(y - (\alpha + \beta \times x)\big)^2]$$

Differentiate first with respect to α.

$$\frac{\partial S}{\partial \alpha} = -2 \times sum[(x,y) : X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times \big(y - (\alpha + \beta \times x)\big)]$$

At the minimum,

$$\frac{\partial S}{\partial \alpha} = 0$$

$$sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times \big(y - (\alpha + \beta \times x)\big)] = 0$$

$$sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times y] =$$

$$sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times \alpha] +$$

$$sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times \beta \times x]$$

$$\bar{y} = \alpha + \beta \times \bar{x}$$

Substitute this back into the original expression for S and differentiate with respect to β.

$$S = sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[ts] \cdot tr[(x,y)] \times \big(y - (\bar{y} - \beta \times \bar{x} + \beta \times x)\big)^2]$$

$$S = sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[ts] \cdot tr[(x,y)] \times \big((y - \bar{y}) + \beta \times (x - \bar{x})\big)^2]$$

$$\frac{\partial S}{\partial \beta} = -2 \times sum[(x,y){:}\, X \times Y \mid (x,y)$$

$$\in dom[tr] \cdot tr[(x,y)] \times \big((y - \bar{y}) + \beta \times (x - \bar{x})\big) \times (x - \bar{x})]$$

At the minimum,

$$\frac{\partial S}{\partial \beta} = 0$$

$$sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times \big((y - \bar{y}) + \beta \times (x - \bar{x})\big) \times (x - \bar{x})] = 0$$

$$sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times (y - \bar{y}) \times (x - \bar{x})] -$$

$$\beta \times sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times (x - \bar{x})^2] = 0$$

So, provided that $sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times (x - \bar{x})^2] \neq 0$, i.e. that there is some variation in the x values in the training set:

$$\beta = \frac{sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times (y - \bar{y}) \times (x - \bar{x})]}{sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[tr] \cdot tr[(x,y)] \times (x - \bar{x})^2]}$$

$$\alpha = \bar{y} - \beta \times \bar{x}$$

There is just one pair of values for α and β that give the lowest sum of squared differences and the formulae give those values.

One possible **model performance measure** that could be used with this model is the Mean Absolute Percentage Error (MAPE). To define this we need to specify a test set:

$$te : (X \times Y) \to \mathbb{N}$$

And now the performance measure itself.

$$MAPE[te, m] = \frac{sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[te] \cdot te[(x,y)] \times \left|\frac{(m[x] - y)}{y}\right|]}{sum[(x,y){:}\, X \times Y \mid (x,y) \in dom[te] \cdot te[(x,y)]]}$$

Example 2: Bayes simple linear regression

The Bayesian approach to simple linear regression requires a model that gives probabilities for particular values, not a point prediction, has a different rule for model learning, and all the other things that follow from those differences.

Once again we have some cases already where we know the value of both variables so that enables our model to learn about the relationship between them. These **data** are our training set.

$X\ ==\ \mathbb{R}$

$Y\ ==\ \mathbb{R}$

$tr : (X \times Y) \rightarrow \mathbb{N}$

Having specified the data involved it is time to specify a model instance and the model family that creates it. A specific **model instance** in this type of regression takes as input a number, representing a possible value for the predictor variable, and it returns a probability density function, giving a probability density for any possible value for the predicted number.

$m :\ X \rightarrow (Y \rightarrow \mathbb{R})$

This model gives the more useful and informative probability distribution for possible values.

The **model family** is a function that creates this type of model. It has three parameters, the inputs to the function, α, β, and σ all of which are real numbers. It returns as output a model instance that makes probability predictions. Here I have called it $pslm$, which is short for probabilistic simple linear model.

$pslm : (\mathbb{R} \times \mathbb{R} \times \mathbb{R}) \rightarrow (X \rightarrow (Y \rightarrow \mathbb{R}))$

The **model family** is specified like this:

$\forall\ \alpha, \beta, \sigma : \mathbb{R}, x : X \cdot pslm[\alpha, \beta, \sigma][x] = N[\alpha + \beta \times x, \sigma^2]$

In words, for all α, β, and σ and all x, where x is of the type X (a real number), the model output for x as input is a Normal distribution with mean equal to $\alpha + \beta \times x$ and variance σ^2.

An alternative form for this model family specification is:

$\forall\ \alpha, \beta, \sigma : \mathbb{R}, x : X \cdot pslm[\alpha, \beta, \sigma][x] = \alpha + \beta \times x + N[0, \sigma^2]$

Where the '+' sign between $\alpha + \beta \times x$ and $N[0, \sigma^2]$ is understood to mean that the $\alpha + \beta \times x$ is added to the value in the domain of the Normal distribution.

The **model learning rule** by which the relationship between the two variables is learned in a Bayesian approach is fundamentally different. Instead of picking one possible combination of values to define the single best model instance, the Bayesian approach is to work out the relative probability of each possible combination of values for those parameters. This is done by taking an initial view of the probabilities of each combination being the best (the prior probability distribution) then updating it in light of the data in the training set to create a revised view of the probabilities of each combination being the best (the posterior probability distribution).

This **model learning rule** can be specified as follows:

$prior, posterior: (\mathbb{R} \times \mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$

$k: \mathbb{R}$

$\forall \alpha, \beta, \sigma: \mathbb{R} \cdot posterior[\alpha, \beta, \sigma] =$

$$k \times prior[\alpha, \beta, \sigma] \times prod[(x, y): X \times Y \mid (x, y) \in dom[tr] \cdot pslm[\alpha, \beta, \sigma][x][y]^{tr[(x,y)]}]$$

Where k is a constant chosen to make the posterior distribution a probability density function, with an area under the curve of 1. The $prod[\ldots]$ part of this is the probability of the actual y values of the training set pairs, assuming each potential combination of α, β, and σ. In other words, what probability would each possible model instance predict for the actual values of y in the training set, given the x values.

The **model learning rule proof** is, again, a bit too complicated to include in this illustration but is just a proof of Bayes' rule applied to probability density functions.

Various alternative **model learning processes** are needed depending on the form of the prior probability distribution. One particular choice of prior probability distribution formulae is known as a conjugate prior, where the form of the posterior probability distribution is the same – with just the parameters changing value. In this case the parameters of the posterior distribution can be calculated directly from the parameters of the prior distribution and the data.

In other cases this kind of calculation is not possible and an approximate, iterative method might be needed.

All these methods require quite advanced mathematics and will not be specified in this article. Their **model learning process proofs** are even more complicated and also will not be included in this article.

A potentially suitable **prediction combination rule** for this distribution of model probabilities is Bayesian Model Averaging, which is the mean of predictions over all possible models, weighted by their posterior probabilities.

One suitable **model performance measure** for this kind of model is Information Gain. This is the information provided by a probabilistic prediction model beyond that provided by some simpler probabilistic prediction model. It is usually averaged across the test set.

Example 3: AR[2] for a time sequence

The AR[2] model family is part of an even larger family and is used to predict the next number of a sequence. The idea of AR[2] is that the next number in the sequence is predicted from the previous two. The rule that does that is a linear function learned by applying regression to earlier numbers in the sequence. The model makes a point prediction for the next value, not a probabilistic one.

The **data** involved in this example consist of a single sequence of numbers representing measurements taken at successive points in time. One sequence provides both the training and test data, with a first section of the sequence used for training and the rest (just the last two values) used for testing.

$ts: seq\ \mathbb{R}$

$\#ts > 4$

In the notation above, the overall time sequence, which consists of Real numbers, is called ts.

We need at least five values in the sequence in order to learn the linear rule. In practice we would like considerably more terms in the sequence.

A specific **model instance** in this type of regression takes as input two numbers, being the previous two numbers in the sequence, and returns a predicted value for the next number in the sequence.

$ar2m : (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$

The **model family** is a function that creates this type of model. It has three parameters, the inputs to the model family function, α, β, and γ, all of which are real numbers. It returns as output a model instance that makes predictions. Here is the name and type.

$ar2mf : (\mathbb{R} \times \mathbb{R} \times \mathbb{R}) \rightarrow ((\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R})$

The **model family** is specified like this:

$$\forall\, \alpha, \beta, \gamma : \mathbb{R}, t_1, t_2 : \mathbb{R} \cdot ar2mf[\alpha, \beta, \gamma] = ar2m \ \wedge$$
$$ar2m[t_1, t_2] = \ \alpha + \beta \times t_1 + \gamma \times t_2$$

In words, for all α, β, and γ, and all t_1 and t_2 which are Real numbers, the model output for t_1 and t_2 as input is equal to $\alpha + \beta t_1 + \gamma t_2$. Instances of this model family are models with specific values for α, β, and γ. For example,

$$\forall\, t_1, t_2 : \mathbb{R} \cdot ar2mf[1,2,3][t_1, t_2] = 1 + 2 \times t_1 + 3 \times t_2$$

The rule by which the relationship between the two previous values and the next encoded into the model is conventionally one that finds values for α, β, and γ that minimise the total of the squares of the differences between the model's predictions and the actual data, summed across the time sequence. This is the OLS criterion again.

This **model learning rule** can be specified as follows:

$\alpha_{OLS}, \beta_{OLS}, \gamma_{OLS} : \ \mathbb{R}$

$$\neg\exists(\alpha', \beta', \gamma') : \mathbb{R} \mid sum[i : \mathbb{N} \mid i \in 3..\#ts \cdot \left(ts[i] - (\alpha' + \beta' ts[i-2] + \gamma' ts[i-1])\right)^2]$$
$$< sum[i : \mathbb{N} \mid i \in 3..\#ts \cdot \left(ts[i] - (\alpha_{OLS} + \beta_{OLS} ts[i-2] + \gamma_{OLS} ts[i-1])\right)^2]$$

As with Example 1, the simple linear model, this says that there is no other combination of parameter values that is better. That there is only one such combination of values can be proved as shown in Example 1. The model learning rule proof, model learning process and its proof, and typical model performance measure are all as for the simple linear model in Example 1.

References

Spivey, J.M. (1989). The Z Notation. Prentice-Hall, Englewood Cliffs, NJ. Available online at: http://spivey.oriel.ox.ac.uk/mike/zrm/zrm.pdf

Appendix

In defining the criteria for model ‘fitting’ I represented the training set as a bag of X and Y values. A slightly more familiar looking approach is to represent the training set with a sequence of X values and a sequence of Y values. This way the familiar index i is involved.

Here is the OLS criterion done in this way, with x and y taking the place of ts:

$x : seq\ X$

$y : seq\ Y$

$mf : Q \to (X \to Y)$

$q : Q$

$$\neg\exists\ q' : Q \mid sum[i : \mathbb{N} \mid i \in 1..\#x \cdot (y[i] - mf[q'][x[i]])^2] < sum[i : \mathbb{N} \mid i \in 1..\#x \cdot (t[i] - mf[q][x[i]])^2]$$

Another conventional refinement is to use the $argmin$ function, which returns the set of arguments giving the minimum value for the function. In this case that would be something like this:

$$q \in argmin[q' : Q \cdot sum[i : \mathbb{N} \mid i \in 1..\#x \cdot (t[i] - mf[q'][x[i]])^2]]$$

And here is the MLE criterion:

$x : seq\ X$

$y : seq\ Y$

$mf : Q \to (X \to (Y \to \mathbb{R}))$

$q : Q$

$$\neg\exists\ q' : Q \mid prod[i : \mathbb{N} \mid i \in 1..\#x \cdot mf[q'][x[i]][y[i]] > prod[i : \mathbb{N} \mid i \in 1..\#x \cdot mf[q][x][y]]$$

Or using $argmax$,

$$q \in argmax[q' : Q \cdot prod[i : \mathbb{N} \mid i \in 1..\#x \cdot mf[q'][x[i]][y[i]]]$$

And, finally, the Bayesian approach:

$x : seq\ X$

$y : seq\ Y$

$mf : Q \to (X \to (Y \to \mathbb{R}))$

$qs : \mathbb{P}\ Q$

$prior : Q \to \mathbb{R}$

$post : Q \to \mathbb{R}$

$dom[prior] = qs$

$dom[post] = qs$

$$\forall\ q : Q \mid q \in qs \cdot post[q] = prior[q] \times \frac{prod[i : \mathbb{N} \mid i \in 1..\#x \cdot mf[q][x[i]][y[i]]]}{sum[q' : Q \mid q' \in qs \cdot prod[i : \mathbb{N} \mid i \in 1..\#x \cdot mf[q'][x][y]]]}$$