Biophysics 101: Modeling Group
Modeling Software Documentation

Alex Lupsasca

December 2009

Introduction

The Modeling Group's task, as part of the class project, was to produce software that would take as input information about a person's genotype, and that would then output a prediction about the person's phenotype.

This initial idea later evolved in several ways. Perhaps the most important one is the form taken by the output, which is actually not a prediction, but rather a *model* for making predictions. The major advantage of this approach is that these models need only be generated once (when the data is first obtained), and then other software such as Trait-o-Matic can call upon the models to make trait predictions based on available genotypic information. The upshot of all this is the high computational efficiency and modularity of the code.

Underlying Model

Pre-existing models in the literature

The Biology Group provided us with a paper[1] which describes a procedure for producing a model that predicts an individual's eye color from their genotype, more specifically from 6 SNP's that were identified as relevant. This was successfully tested on a sample of 6000+ people: a set of 3804 people were used to establish the model, which was then tested on 2364 people. The overall accuracy of the predictions was more than 90%.

This was done using the method of ordinal regression, which is often used when the prediction to be made is categorical (i.e. "blue" or "intermediate" or "brown") and the outcomes are ordered, that is, each category is assigned a numerical threshold on a continuous scale: for instance, eye color can be quantified by a number in the interval $[0, 1]$, with "blue" corresponding to the sub-interval $[0, a]$, "intermediate" to $[a, b]$ and brown to $[b, 1]$, where $0 < a < b < 1$.

The paper in question used precisely such a model. They let eye color y be one of three ordinal levels ("blue", "intermediate" and "brown") which are determined by the genotype $\vec{x}$ of k SNPs, $\vec{x} = (x_1, x_2, \ldots, x_k)$. Then, letting π_1, π_2 and π_3 denote the proportion of the population (i.e. the probability) with "blue", "intermediate" and "brown", respectively, the ordinal regression they used was

$$\text{logit}(\Pr(y \leq \text{blue} \mid \vec{x})) = \ln\left(\frac{\pi_1}{1-\pi_1}\right) = \alpha_1 + \sum_{i=1}^{k} \beta_i x_i;$$

$$\text{logit}(\Pr(y \leq \text{inter} \mid \vec{x})) = \ln\left(\frac{\pi_1 + \pi_2}{1-(\pi_1+\pi_2)}\right) = \alpha_2 + \sum_{i=1}^{k} \beta_i x_i,$$

[1] *Eye color and the prediction of complex phenotypes from genotypes*, **Current Biology** Vol 19 No 5 R192

where α and β were infered from the model-building set. Note that the logit function, defined by

$$\begin{aligned}\text{logit} &: [0,1] \to [-\infty, \infty] \\ \text{logit} &: \pi \longmapsto \ln\left(\frac{\pi}{1-\pi}\right)\end{aligned}$$

takes a probability between 0 and 1 to a continuous real variable z, which can be arbitrarily small or large.

Then the eye color of each individual in the model-verification set was predicted based on their genotype and the derived α and β:

$$\begin{aligned}\pi_1 &= \frac{\exp\left(\alpha_1 + \sum_{i=1}^{k} \beta_i x_i\right)}{1 + \exp\left(\alpha_1 + \sum_{i=1}^{k} \beta_i x_i\right)}; \\ \pi_2 &= \frac{\exp\left(\alpha_2 + \sum_{i=1}^{k} \beta_i x_i\right)}{1 + \exp\left(\alpha_2 + \sum_{i=1}^{k} \beta_i x_i\right)} - \pi_1; \\ \pi_3 &= 1 - (\pi_1 + \pi_2).\end{aligned}$$

This type of "logit" regression, or logistic regression, is often used in the literature[2] because of the following nice features: first it is linear (and therefore fast to compute); next, it makes use of a variable

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k,$$

whose origin is intuitively clear: each SNP (or environmental factor, or really any factor whatsoever) x_i is assigned a relative importance β_i and the influences of

[2]For instance, in *Logistic Regression Model of Binary Disease Trait for Case-Control Study Considering Interactions between SNPs and Environments*, `http://www.jsbi.org/journal/GIW04/GIW04P106.pdf`.

all factors are summed over in a single variable. This variable can take any value from negative to positive infinity, which allows for a large spread in the observed data; this is where the inverse logit function comes in:

$$\text{logit}^{-1} : [-\infty, \infty] \to [0, 1]$$
$$\text{logit}^{-1} : z \longmapsto \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

Thus, the inverse logit function transforms the variable into a probability between 0 and 1. Moreover, in doing so, it exponentiates z, thereby introducing a multiplicative effect:

$$e^z = e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k} = e^{\beta_0} e^{\beta_1 x_1} e^{\beta_2 x_2} \cdots e^{\beta_k x_k}$$

In conclusion, then, logistic regression is a fast, simple regression model which incorporates additive and multiplicative effects, and which has already been shown to predict with great accuracy eye color. As such, we decided to base our own work on this model.

Our model

Based off existing work, we consider a number n of phenotypic traits y_1 through y_n, where each y_i can take on a discrete value in a pre-determined set. For instance, when looking at eye color, we could work with a set $Y_i = \{0, 1, 2\}$, where the number quantify the colors "blue", "green" and "brown", respectively. This easily extends to other discrete traits. In the continuous case, on the other hand, this necessitates setting a resolution for discrete quantization: in the case

of height, for instance, we could have a set

$$Y_i = \{[160, 170], [170, 180], [180, 190], [190, 200]\}$$

in which each element actually encompasses a range of heights. In this example, the resolution is quite low (10 centimeters), but the level of granularity could be increased by reducing the interval size of the discrete quantization (say, to 5 centimeters). Note that this would result in the addition of new elements to the set (here, this would double the size from 4 to 8 possibilities) and this would incur a computational cost when the regression is run. Moreover, increased resolution always translates into lower accuracy, and so finding the balance requires human judgement.

In any case, this results in a vector

$$\vec{y} = (y_1, y_2, \ldots, y_n)$$

which fully encodes the phenotypic information. In our work so far, we have only conidered one trait at a type, and so our vector has actually been a single variable $\vec{y} = y$. There is no particular reason why we should restrict ourselves to this, other than practicality in running tests. Increasing the number of traits to $n >$ is exactly equivalent to running n separate one-trait regressions, and so nothing is lost.

In our model, we view an individual's genotypic information as a vector

$$\vec{x} = (x_1, x_2, \ldots, x_k)$$

where each x_i is a binary piece of information which stands for either "SNP i is present" ($x_i = 1$) or "SNP i is absent" ($x_i = 0$). Note that this is still absolutely

general, and in no way limits the power of the model: if a SNP were to have more than one possible type (say AA, or AG, etc.) then it could be assigned more than one x_i. More precisely, a factor (be it a SNP, or an environmental factor) which can take on m values will be assigned n of the binary x_i, where n is the least power of 2 which exceeds m:

$$n = \min\{j \mid 2^j \geq m\}.$$

In fact, the same can be done with the phenotype, in which case the y_i also represent binary information.

Next, for every individual in a model-building set, we take in their genome vector $\vec{x}$ and associate to it an observational data vector

$$\vec{d} = (\vec{x}, \vec{y}) = (x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_n).$$

The observed data of all the individuals is then gathered into the following array:

	x_1	x_2	$\cdots$	x_k
y_1	$N_{1,1}$	$N_{1,2}$	$\cdots$	$N_{1,k}$
y_2	$N_{2,1}$	$N_{2,2}$	$\cdots$	$N_{2,k}$
$\vdots$			$\vdots$	
y_k	$N_{k,1}$	$N_{k,2}$	$\cdots$	$N_{k,k}$

Here, each $N_{i,j}$ counts the number of individuals who have SNP j and exhibit trait i, that is for which $y_i x_j = 1$. This can be put in matrix form: the data

matrix D is

$$D_{ij} = \begin{pmatrix} N_{1,1} & \cdots & N_{1,k} \\ \vdots & \ddots & \vdots \\ N_{k,1} & \cdots & N_{k,k} \end{pmatrix}$$

Dividing through by the total number T of people in the model-building set, this results in a probability matrix P:

$$P_{ij} = \frac{D_{ij}}{T} = \begin{pmatrix} \Pr(1,1) & \cdots & \Pr(1,k) \\ \vdots & \ddots & \vdots \\ \Pr(k,1) & \cdots & \Pr(k,k) \end{pmatrix}$$

The $(i,j)^{\text{th}}$ entry is $\Pr(i,j) = \frac{N_{i,j}}{T}$. It corresponds to the conditional probability of someone exhibiting trait i given that they have SNP j; note that by construction, for all i and j, $0 \leq \Pr_{i,j} \leq 1$, as desired.

Question: why not stop here?

Indeed, suppose that for some i and j, $\Pr(i,j) = 1$. Then given someone with $x_j = 1$, that is, with the SNP j, we can infer with great probability that they will exhibit trait j. Or, suppose that we are given a genotype $vecx$ which is the exactly the same as a very large number (say, 100) of genotypes in our model-building set. Then in that case, we can directly predict the probability of that person having trait j simply by looking at the proportion of individuals with the same genotype $\vec{x}$ who exhibit trait j.

But then we have accomplished nothing special: taking this to the extreme, we could just look at the entire sample population in our model-building set and say that $\sum_i \Pr(i,j)$ is the general probability for someone in the wider population (of which we assume our model-building set to be a representative sample) to

exhibit trait j. This is not very useful, especially since we can do much better...

Indeed, the reason we do not want to stop here is that so far, no "learning" has occurred. All we have done is look at data and use it to produce banal probabilities; however, we do not know how these probabilities interact...

Moreover, if someone comes along with a genotype $\vec{x}$ which is different from all the genotypes we have seen before, we do not have a good basis for making predictions about their phenotype. However, this could be remedied, if only we could infer from our data *how* the SNPs interact.

This is where logistic regression comes in...

We now proceed with the simplified case $n = 1$ of one trait, noting again that this does not in any way reduce the power of generality of the model (it simply makes it necessary to run the model n times to make predictions on each of the n traits in $\vec{y}$). In this case, the matrix P_{ij} is just a row vector

$$\vec{P} = \big(\Pr 1, \Pr 2, \ldots, \Pr k\big) = \big(\Pr(y \mid x_1), \Pr(y \mid x_2), \ldots, \Pr(y \mid x_k)\big).$$

We then apply the logit function to obtain continuous variables with a much larger spread, which will make the regression much more effective:

$$\text{logit } \vec{P} = \big(\text{logit}(\Pr 1), \text{logit}(\Pr 2), \ldots, \text{logit}(\Pr k)\big).$$

This allows us to run a linear regression to find the relative importance of each x_i in the determination of trait y. That is, the regression yields the "relative importance" factors β_i in

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k.$$

where z is a continuous variable. This regression process is well-known and quite standard; for this reason, it is not described here but can easily be found online or in textbooks. Basically, it amounts to plotting the points x_i and trying to fit a line through then...

As such, taking the inverse logit of the constants β, we obtain a list of inferred probabilities which tell us how likely it is to exhibit trait y given any genotyp $\vec{x}$. Note that these probabilities *have been learned*. They take the form of a vector

$$\vec{I} = \big(\pi_1, \pi_2, \ldots, \pi_k\big),$$

where for each i, $\pi_i = \text{logit}^{-1}(\beta_i) = \frac{1}{1+\exp(-\beta_i)}$. Finally, the likelihood of exhibiting trait y given genotype $\vec{x}$ is just the dot product

$$\vec{x} \cdot \vec{I} = \pi_1 x_1 + \pi_2 x_2 + \ldots + \pi_k x_k.$$

Software Implementation

Overview

Our software implementation model fulfils, at least partially, the initial goal we had set for ourselves: given a model-building set of individuals, it generates the vector $\vec{I} = (\pi_1, \pi_2, \ldots, \pi_k)$ of *learned probabilities* and outputs a model in a form which is understood by Trait-o-Matic (as per the Infrastructure Group's specifications). In turn, *that* model takes in an arbitrary genotype $\vec{x}$ and predicts the likelihood $\vec{x} \cdot \vec{I}$ of that individual expressing phenotype y.

Description of the code

Our implementation was done in Python, which is a very flexible and powerful high-level language. It was chosen for its ease of use and modularity – indeed, we tried to make our code highly readable (by including plenty of comments) and easily modifiable (by implementing different aspects of the model in different modules).

As such, our program consists of three parts: `logregoutput.py`, `dataread.py` and `ols.py`. The function of each of these parts will now be described; as to the code itself, it shall not be examined in this documentation (we reiterate that is heavily commented and should be easy to read, and understand).

The observed data (i.e. the set of all vectors $\vec{d}$ corresponding to the individuals in the model-building set) is written into a simple `.csv` file. This stands for "comma-separated values" and is the simplest format for storing arrays of information such as the one from the preceding section. It is also recognized by Microsoft Excel, which makes it easy to share and modify. A sample data set `testdata.csv` would thus take the following form:

```
snp1,snp2,snp3,phenotype
1,1,1,1
1,1,0,1
1,1,0,1
1,1,1,1
1,1,1,1
...
```

where `snp1` is the name of SNP 1 and represents the binary value of x_1, and so on. Similarly, `phenotype` is the name of the binary trait y under consideration.

Now, the main routine is `logregoutput.py`. It can be called at the command line with a data file such as the above as argument: for instance, provided that a file `testdata.csv` of the above type is located in the same directory as the program, one would run the software with the command

```
python logregoutput.py testdata.csv
```

The program will then call the routine `dataread.py` to read in the data and produce the previously described array, together with the associated probability vector $\vec{P}$. `logregoutput.py` will then proceed to compute the logit of the probabilities logit $\vec{P}$, and then will call on `ols.py` to actually perform the linear regression. Note that this file was not written by us; it is a standard library that is made freely available online.

Finally, `logregoutput.py` will print to screen the following output:

```
# rsid:snp1
# rsid:snp2
# rsid:snp3

##### This is an automatically generated model file created by logistic
##### regression on the SNPs listed above.

from numpy import *
from sys import argv

genotypes = argv[1:]

genotype_list = [1] + [int(genotype) for genotype in genotypes]
```

```
genotype_array = array(genotype_list)

##### PVALS GO HERE #####
pvals = array([ 0.00589634,  0.00456349,  0.27948336,  0.0756931 ])

val = -1*inner(genotype_array, pvals)
risk = 1/(1+exp(val))
print risk
```

Output and interface with Trait-o-Matic

An output of the type seen above is itself a model. Its format obeys the specifications passed on to us by the Infrastructure Group, and is understandable by Trait-o-Matic. Note that the name of SNP 1, `snp1` becomes an rsid tag for Trait-o-Matic; this allows for the automation of the genomic information retrieval process.

Another advantadge is that this type of output can also be easily generated by humans. Furthermore, should a human wish to modify the output of our model (in the case that our automated output suffers from an unforeseen problem, for instance), then they could easily do so in a text editor.

Finally, the model can be run either in Trait-o-Matic, or directly at the command line. Assuming that the above output model was stored in a file `model.py`, we could compute the likelihood of an individual expressing phenotypic trait y as follows: supposing that they have genotype $\vec{x} = (1, 0, 1)$ (that is, they have SNPs 1 and 3, but not SNP 2), we would call

```
python model.py 1 0 1
```

This would output something of the form

```
0.521524920396
```

which indicates a probability of 52% of expressing the trait.

Further Developments

Please refer to the Biophysics 101 Wiki's documention page, where Zach compiled the discussions of the Modeling Group.