A Framework and Application for Efficient Analysis of Peptide Libraries

Eric Riemer Hare

November 12, 2013

1 Introduction

Peptide libraries have immensely important biological and medical applications[REF]. However, analysis of properties of these libraries is a challenging task. With 20 potential amino acids and a peptide length k, there are 20^k distinct peptides that can be encoded. The situation is further complicated by the fact that different sets of codons can encode the same amino acid. This phenomenon is illustrated in Figure 1. As this wheel represents the possibilities for only one of the k positions in the peptide sequence, its clear that this problem becomes exponentially more complex as the peptide length increases.

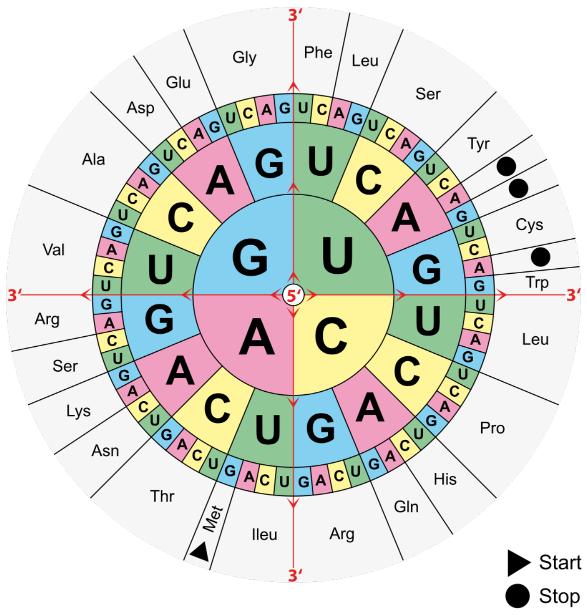

Figure 1: Codon wheel illustrating the sequences of codons leading to each amino acid, as well as start and stop tags.

In order to assess the quality of a particular library of peptides, some building blocks are needed in order to proceed. Specifically, treating the presence of each amino acid in a peptide as independent of any other amino acid in that sequence, we can define the joint distribution:

$$P(A_1 = a_1, A_2 = a_2, ..., A_k = a_k) = P(A_1 = a_1)P(A_2 = a_2) \cdots P(A_k = a_k)$$

A_1 through A_k represent random variables associated with the amino acid at each of positions one through k in the peptide sequence. The need for a clean and consise representation of jointly discrete random variables is clear. I will now introduce the software-based building blocks used to represent peptides.

2 Backend

There are two major components that represent the backend of the application and analysis. The modular nature of these components are convenient for extensions to this work, both in the analysis of peptide libraries, and any other application in which jointly discrete random variables are useful as a representation of particular phenomena.

2.1 discreteRV

Dr. Andreas Buja, professor of Statistics at the University of Pennsylvania, created a set of R functions implementing discrete random variables[REF]. These functions were compiled and documented in a package called discreteRV. discreteRV is available for download on the Comprehensive R Archive Network (CRAN)[REF].

The functions of discreteRV are organized into two logical areas, termed probabilities and simulations.

2.1.1 Probabilities

The centerpiece of discreteRV is a set of functions to create and manipulate discrete random variables. A list of these functions and brief discriptions of their functionality is available in table 1. A random variable is defined through the use of the make.RV function. make.RV accepts a vector of probabilities and a vector of outcome values, and returns an RV object.

```
make.RV(vals = 1:6, probs = rep("1/6", times = 6))


## random variable with 6 outcomes
##
##   1   2   3   4   5   6
## 1/6 1/6 1/6 1/6 1/6 1/6
```

Name	Arguments	Description

Table 1: List of the probability functions contained in discreteRV.

The syntactic structure of the included functions lends itself both to a natural presentation in an introductory probability course, as well as more advanced modeling of discrete random variables. The structure of an RV object is illustrated in Figure 2. The object is constructed by setting a standard R vector object to the possible values that the random variable can take (the sample space). It is preferred, though not required, that these be encoded as integers, since this allows for expected values, variances, and other measures of distributional tendency to be computed. This vector of outcomes is then named by the respective probability of each outcome. The probability can be encoded as a string, such as “1/6”, if this aids in readability, but the string must be coercable to a numeric.

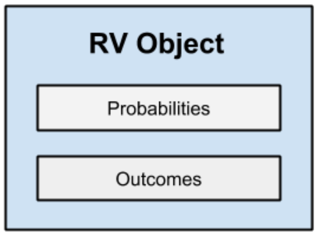

Figure 2: Structure of an RV object

The choice to encode the probabilities in the names of the vector may seem counterintuitive. However, it is this choice which allows for the familiar syntax employed by introductory statistics courses and textbooks to be seamlessly replicated. Consider, for instance, an RV object “X” which we will use to represent a single roll of a fair die.

```
X <- make.RV(1:6, rep("1/6", 6))
X


## random variable with 6 outcomes
##
##   1   2   3   4   5   6
## 1/6 1/6 1/6 1/6 1/6 1/6
```

Note that although the print method does not illustrate the inherent structure of the object, the probabilities (1/6, for each of the 6 outcomes of the die roll) are actually stored in the names of the object "X".

```
names(X)


## [1] "1/6" "1/6" "1/6" "1/6" "1/6" "1/6"
```

By storing the outcomes as the principle component of the object X, we can now make a number of probability statements in R. For instance, we can ask what the probability of obtaining a roll greater than 1 is by using the code *P(X ¿ 1)*. R will check which values in the vector X are greater than 1. In this case, these are the outcomes 2, 3, 4, 5, and 6. Hence, R will return TRUE for these elements of X, and then we can encode a function P to compute the probability of this occurrence by simply summing over the probability values stored in the names of these particular outcomes. Likewise, we can make slightly more complicated probability statements such as *P(X ¿ 5 — X == 1)*.

Aside from moments and probability statements, discreteRV includes a powerful set of functions used to create joint probability distributions. Once again letting X be a random variable representing a single die roll, we can use the *multN* function to compute the probability mass function of n trials of X.

```
library(xtable)

XX <- multN(X, fractions = TRUE)


## Loading required package:  MASS


xtable(t(data.frame(Outcome = as.character(XX), Probability = names(XX))))
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Outcome	1.1	1.2	1.3	1.4	1.5	1.6	2.1	2.2	2.3	2.4	2.5	2.6	3.1	3.2	3.3
Probability	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36

2.1.2 Simulation

2.2 peptider

2.2.1 Expected Coverage

2.2.2 Relative Efficiency

2.2.3 Functional Diversity

2.2.4 Peptide Inclusion

2.2.5 Neighborhoods

2.2.6 Custom Schemes

3 Frontend

3.1 PeLiCa

3.1.1 Overview

3.1.2 Shiny

3.1.3 User Interface

3.1.4 Features

4 Further Work