A Framework and Application for Efficient Analysis of Peptide Libraries

Eric Riemer Hare

January 25, 2014

My creative component consists of three separate papers, each covering a different component of the overall project I worked on. The layering of the three components is illustrated in Figure 1. The structure of this document is organized in the same manner, beginning with discreteRV, continuing with peptider, and ending with PeLiCa.

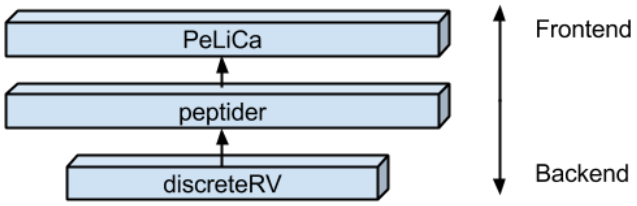


Figure 1: The three components, or layers, of my project.

Manipulation of Discrete Random Variables with discreteRV

by Eric Riemer Hare, Andreas Buja, and Heike Hofmann

Abstract A major issue in statistics education is the sometimes large disparity between the mathematical and theoretical coursework, and the computational coursework. discreteRV is an R package for manipulation of discrete random variables which uses clean and familiar syntax similar to that which is found in introductory probability courses. It is simple enough for those less experienced with statistical programming, but has more advanced features which are suitable for a large number of more complex applications. In this paper, I introduce and motivate discreteRV, describe its functionality, and provide reproducible examples illustrating its use.

Introduction

One of the primary hurdles in teaching probability courses in an undergraduate setting is the missing link between theoretical statements from textbooks and lectures, and the notation used in statistical software required in more and more classes. Depending on the background of the student, this missing link can manifest itself in a couple of different ways. Some students will master the theoretical concepts and notation, but struggle in a computing environment. Others will feel very comfortable with statistical programming, but sometimes struggle to translate that back to the classroom probability setting.

discreteRV is an attempt to bridge this gap. It provides a comprehensive set of functions to create, manipulate, and simulate from discrete random variables. It uses syntax which in most cases closely matches probability textbooks to allow for a more seamless connection between a probability classroom setting and the use of statistical software. **discreteRV** is available for download on the Comprehensive R Archive Network (CRAN).

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

Probabilities

discreteRV includes a suite of functions to create, manipulate, and compute distributional quantities for the random variables defined. A list of these functions and brief discriptions of their functionality is available in Table 1. In Table 2, the notational similarities between **discreteRV** and the commonly used book George Casella (2001) is shown.

Name	Description
Creation	
`as.RV`	Turn a probability vector with possible outcome values in the names() attribute into a random variable
`make.RV`	Make a random variable consisting of possible outcome values and their probabilities or odds
Manipulation	
`margins`	Marginal distribution of a joint random variable
`mult`	Joint probability mass function of random variables X and Y
`multN`	Probability mass function of X^n
`SofI`	Sum of independent random variables
`SofIID`	Sum of independent identically distributed random variables
Probabilities	
`E`	Expected value of a random variable
`KURT`	Kurtosis of a random variable
`P`	Calculate probabilities of events
`probs`	Probability mass function of random variable X
`SD`	Standard deviation of a random variable
`SKEW`	Skewness of a random variable
`V`	Variance of a random variable
Methods	
`plot.RV`	Plot a random variable of class RV
`print.RV`	Print a random variable of class RV
`qqnorm.RV`	Normal quantile plot for RVs to answer the question how close to normal it is

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

discreteRV	Casella and Berger
`E(X)`	E(X)
`P(X == x)`	P(X = x)
`probs(X)`	f(x)
`SD(X)`	sd(X)
`V(X)`	var(X)

Table 2: Probability functions in discreteRV and their corresponding syntax in introductory statistics courses.

Creating random variables

The centerpiece of **discreteRV** is a set of functions to create and manipulate discrete random variables. As per Chris Wild (1999), a random variable is a theoretical construct representing the value of an outcome of a random experiment. A discrete random variable is a random variable that can take on a countable and finite set of values. Discrete random variables are associated with probability mass functions, which map the set of possible outcomes of the random experiement to probabilities which must sum to one. Throughout this document, I will work with a simple example of a discrete random variable representing the value of a roll of a fair die. Formally, we can define such a random variable and its probability mass function as follows:

Let X be a random variable representing a single roll of a fair die. Then,

$$f(x) = P(X = x) = \begin{cases} \frac{1}{6} & x \in 1,2,3,4,5,6 \\ 0 & \text{otherwise} \end{cases}$$

In **discreteRV**, a discrete 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. Consider our example of an RV object `X` which we will use to represent a single roll of a fair die. The code to create such a random variable is as follows:

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

Structure

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 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 to compute expected values, variances, and other distributional properties. 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.

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.

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"
```

Probabilities

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 $P(X > 5 \cup X = 1)$.

Several other distributional quantities are computable, including the expected value and the variance of a random variable. As in notation from probability courses, expected values can be found with the `E` function. To compute the expected value for a single roll of a fair die, we run the code `E(X)`.

Joint Distributions

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. Table 3 gives the first eight outcomes for $n = 2$, and Table 4 gives an the first eight outcomes for $n = 3$. Notice again that the probabilities have been coerced into fractions for readability. Notice also that the outcomes are encoded by the outcomes on each trial separated by a period.

Outcome	1.1	1.2	1.3	1.4	1.5	1.6	2.1	2.2
Probability	1/36	1/36	1/36	1/36	1/36	1/36	1/36	1/36

Table 3: First eight Outcomes and their associated Probabilities for a variable representing two independent rolls of a die.

discreteRV also includes functions to compute the sum of independent random variables. If the variables are identically distributed, the `SofIID` function can be used to compute probabilities for

Outcome	1.1.1	1.1.2	1.1.3	1.1.4	1.1.5	1.1.6	1.2.1	1.2.2
Probability	1/216	1/216	1/216	1/216	1/216	1/216	1/216	1/216

Table 4: First eight Outcomes and their associated Probabilities for a variable representing three independent rolls of a die.

the sum of n independent realizations of the random variable. In our fair die example, `SofIID(X,2)` would create a random variable object with the representation given in 5

Outcome	2	3	4	5	6	7	8	9	10	11	12
Probability	1/36	1/18	1/12	1/9	5/36	1/6	5/36	1/9	1/12	1/18	1/36

Table 5: Outcomes and their associated Probabilities for a variable representing the sum of two independent rolls of a die.

Plotting

discreteRV includes a `plot` method for random variable objects so that a visualization of the outcomes and probabilities can be made simply by calling `plot(X)`. The result of plotting the random variable representing a fair die is given in Figure 1. The x axis includes all outcomes, and the y axis includes the probabilities of each particular outcome. The result of plotting a random variable representing the sum of two independent rolls of a die is given in Figure 2. The result of plotting a random variable representing the sum of 100 independent rolls of a die is given in Figure 3.

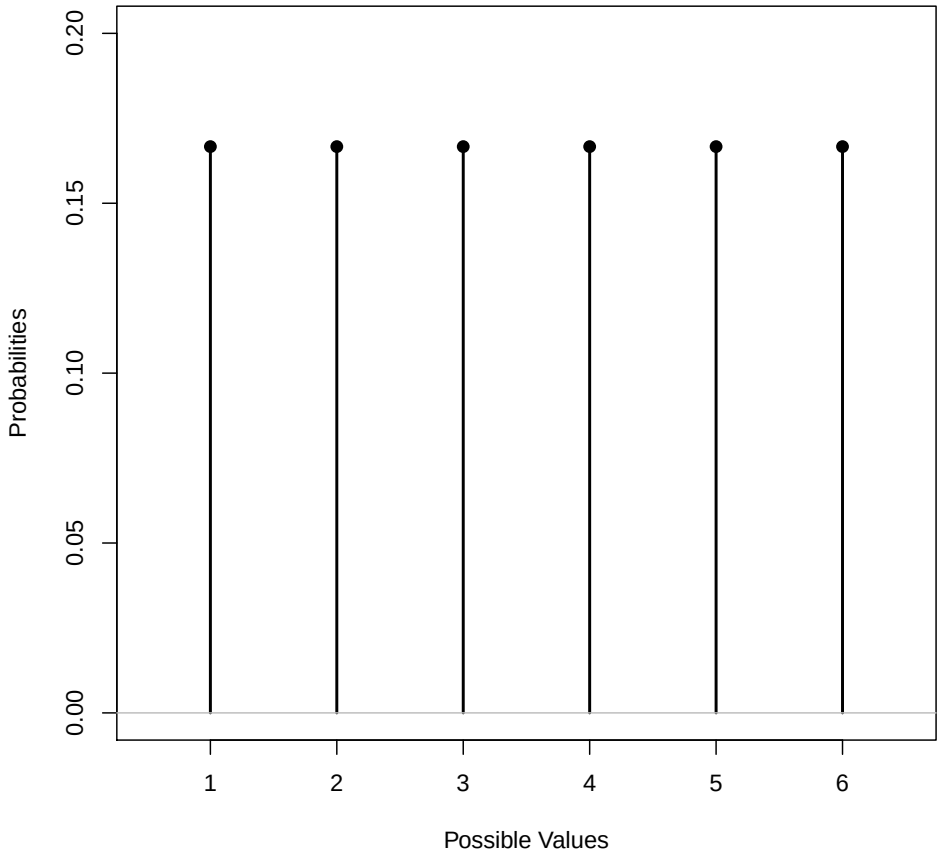


Figure 1: Plot method called on a fair die random variable.

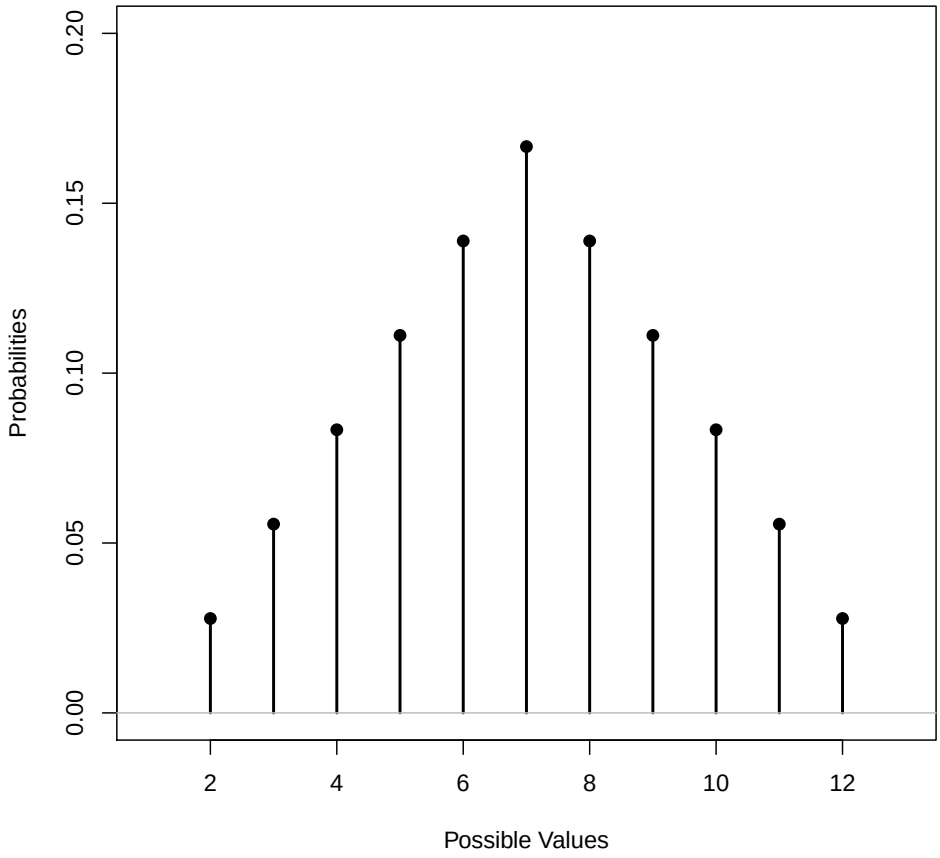


Figure 2: Plot method called on a sum of two fair die random variable.

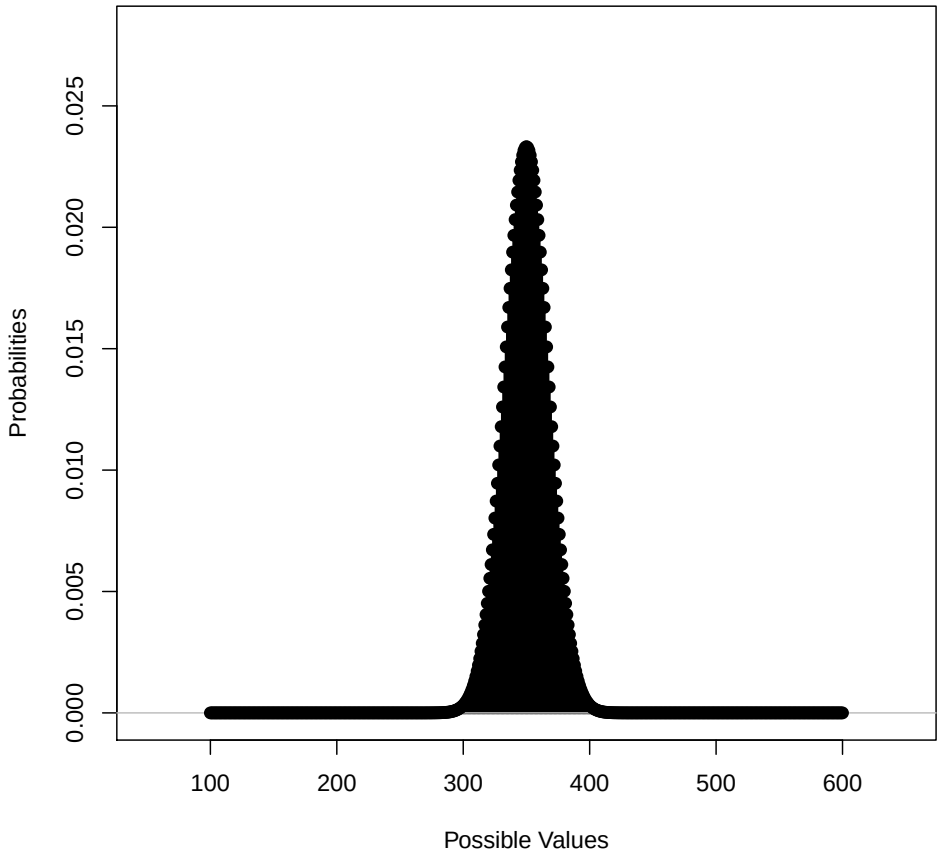


Figure 3: Plot method called on a sum of 100 fair die random variable.

In addition to a plotting method, there is also a method for `qqnorm` to allow assessment of normality for random variable objects, as displayed in Figure 4.

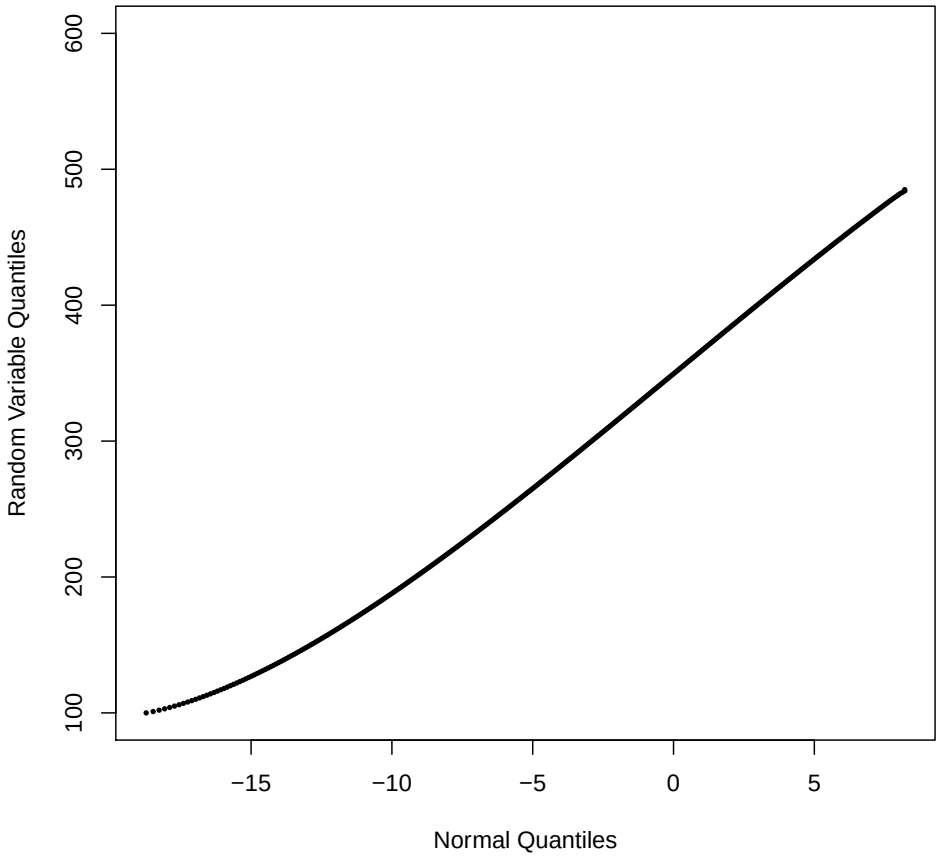


Figure 4: qqnorm method called on a sum of 100 fair die random variable.

Simulation

discreteRV also includes a set of functions to simulate trials from a random variable. A list of these functions and brief discriptions of their functionality is available in Table 6.

Name	Description
plot.RVsim	Plot a simulated random vector
Prop	Proportion of an event observed in a vector of simulated trials
props	Proportions of observed outcomes in one or more vectors of simulated trials
rsim	Simulate n independent trials from a random variable X
skewSim	Skew of the empirical distribution of simulated data

Table 6: List of the simulation functions contained in discreteRV.

Creation

Creating a simulated random vector is done by using the `rsim` function. `rsim` accepts a parameter n representing the number of independent trials to simulate, and a parameter X representing the random variable with which to simulate from. For example, suppose we'd like to simulate ten trials from a fair die. We have already created a random variable object X, so we simply call rsim as follows:

```
X.sim <- rsim(10, X)
X.sim

## 1/6 1/6 1/6 1/6 1/6 1/6 1/6 1/6 1/6 1/6
##   1   4   6   2   6   2   2   1   2   6
## attr(,"RV")
## random variable with 6 outcomes
##
##   1   2   3   4   5   6
## 1/6 1/6 1/6 1/6 1/6 1/6
## attr(,"class")
## [1] "RVsim"
```

The object returned is a vector of simulated values, with a class attribute to contain the random variable that was used for the simulation. If we would like to retrieve only the simulated values and exclude the attached probabilities, we can coerce the object into a vector using R's built-in `as.vector` function.

```
as.vector(X.sim)

##  [1] 1 4 6 2 6 2 2 1 2 6
```

It is also possible to retrieve some quantities from the simulation. We can retrieve the empirical distribution of simulated values with the `props` function. This will return the outcomes from the original random variable object, and the observed proportion of simulated values for each of the outcomes. We can also compute observed proportions of events by using the `Prop` function. Similar to the `P` function for probability computations on random variable objects, `Prop` accepts a variety of logical statements.

```
props(X.sim)

## RV
##   1   2   3   4   5   6
## 0.2 0.4 0.0 0.1 0.0 0.3

Prop(X.sim == 3)

## [1] 0

Prop(X.sim > 3)

## [1] 0.4
```

Conclusion

The power of discreteRV is truly in its simplicity. Because it uses familiar introductory probability syntax, it can allow students who may not be experienced or comfortable with programming to ease into computer-based computations. Nonetheless, discreteRV also includes several powerful functions for analyzing, summing, and combining discrete random variables which can be of use to the experienced programmer.

Bibliography

G. S. Chris Wild. *CHANCE ENCOUNTERS: A First Course in Data Analysis and Inference*. John Wiley & Sons, 1999. [p2]

R. L. B. George Casella. *Statistical Inference*, volume 2. Cengage Learning, 2001. [p1]

Analyzing Peptide Libraries with peptider

by Eric Riemer Hare, Heike Hofmann, and Timo Sieber

Abstract Peptide libraries have important theoretical and practical applications in the fields of biology and medicine. This paper introduces a new R package **peptider** which allows for a statistical analysis of these peptide libraries. Based on the research of Sieber et al., **peptider** implements a suite of functions to calculate functional diversity, relative efficiency, expected coverage, and other measures of peptide library diversity. With flexible support for a number of encoding schemes and library sizes, **peptider** can be used to analyze a wide variety of peptide libraries.

Introduction

Libraries of peptides, or amino acid sequences, have a number of applications in the Biological sciences, from studying protein interactions, to vaccine research (Rodi et al. (1999), Irving et al. (2001)). Despite their importance, little analysis has been done to assess the statistical properties of different peptide libraries.

peptider is a newly-released R package which helps to evaluate many important statistical properties of these libraries. It supports a number of built-in library schemes, including NNN, NNB, NNK, NNS, and trimer schemes. It also allows for easy analysis of user-created custom library schemes. **peptider** makes use of the R package **discreteRV**, which allows for manipulation and analysis of discrete random variables. By treating each amino acid in a peptide as a realization of an independent draw from the pool of all possible amino acids, probabilities for the occurrence of peptides can easily be formulated.

This paper will focus on two distinct functional areas of peptider. The first is Library Diversity, or statistical measures of the quality of the library itself. The second is Peptide Coverage, or how likely the library is to include particularly desired peptides, or peptides that are most similar to desired peptides. Before proceeding to discuss these measures, we will first discuss the built-in library schemes, and how to define custom schemes.

Library Schemes

peptider has several built-in library schemes. The first is the NNN scheme, in which all four bases (Adenine, Guanine, Cytosine, and Thymine) can occur at all three positions in a particular codon, and hence there are 64 possible nucleotides. The second is the NNB scheme, where the first two positions are unrestricted, but the third position can only be three bases, yielding 48 nucleotides. Both NNK and NNS have identical statistical properties in this analysis, with the third position restricted to two bases for a total of 32 nucleotides. Finally, there are trimer-based libraries in which the codons are pre-defined. Each of these scheme definitions can be accessed with the `scheme` function.

```
scheme("NNN")
```

```
##   class    aacids c
## 1     A       SLR 6
## 2     B     AGPTV 4
## 3     C         I 3
## 4     D DEFHKNQYC 2
## 5     E        MW 1
## 6     Z         * 3
```

To build a library of an appropriate scheme, the libscheme function is used. By default, peptides of length one amino acid (k = 1) will be used, but this can be specified.

```
nnk6 <- libscheme("NNK", k = 6)
```

libscheme returns a list containing two elements. The first, data, describes the probability of occurrence of each possible peptide class. The second, info, describes the number of nucleotides, the number of valid nucleotides, and the scheme definition used.

We can also create a custom library scheme by building a data frame of the same format as in Code Example 1. This code creates a custom trimer-based library with peptides of length six.

```
custom <- data.frame(class = c("A", "Z"), aacids = c("SLRAGPTVIDEFHKNQYMW",
    "*"), c = c(1, 0))
custom6 <- libscheme(custom, k = 6)
```

Having created the library of interest, we now turn our attention to assessment of these libraries.

Library Diversity

In this section, we introduce a number of properties which can be used to determine the quality of a given peptide library, and which are computable using peptider.

Functional Diversity

The functional diversity of a library is the overall number of different peptides in the library. Analyzing the peptide sequences directly is complex, so a useful approach is to partition the library into amino acid classes, wherein each amino acid belonging to a particular class has the same number of codon representations as all other amino acids in that class. Letting v represent the number of valid amino acid classes, k represent the number of amino acids in each peptide, b_i represent the number of different peptides in class i, N represent the total size of the peptide library in number of peptides, and p_i represent the probability of peptide class i, then the functional diversity is:

$$D(N,k) = \sum_{i=1}^{v^k} b_i(1 - e^{-Np_i/b_i})$$

To compute this diversity measure in peptider, the `makowski` function can be used.

```
makowski(6, "NNK")

## [1] 0.2918
```

Expected Coverage

The expected coverage of the library is directly related to the diversity of the library. It is the percentage of all possible peptides that the library contains. Letting c represent the number of viable amino acids in the library scheme, the expected coverage is:

$$C(N,k) = D(N,k)/c^k$$

Note that the diversity of the library can range between 0 (for a library with no peptides) and c^k (wherein the library includes every possible peptide). Hence, the coverage must range from 0 to 1. **peptider** includes a function to compute the coverage, which again takes the peptide length and library scheme as parameters. It also accepts a parameter N representing the overall number of peptides in the library.

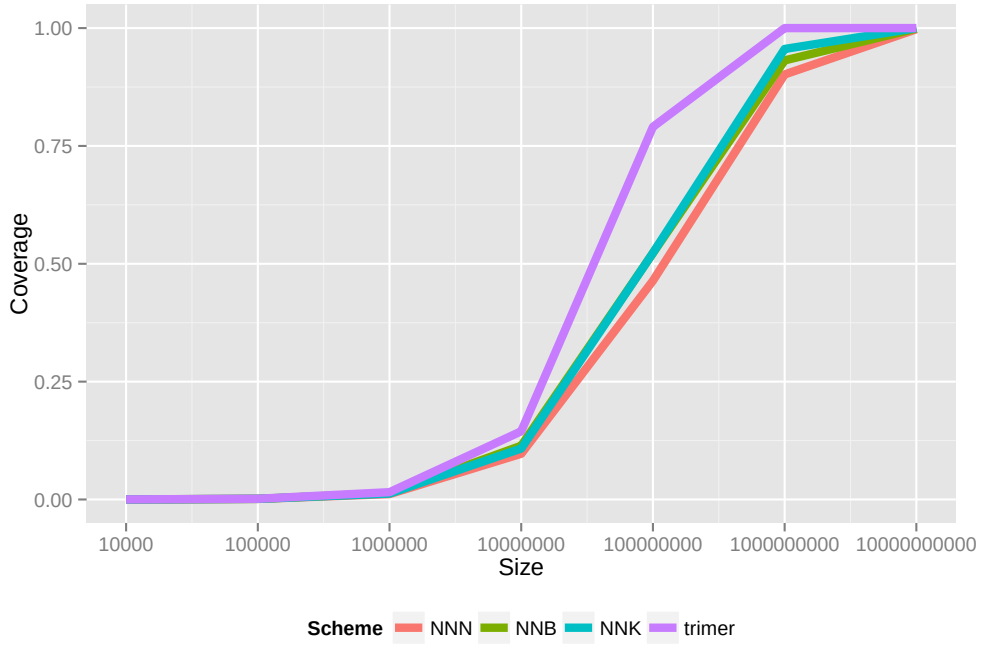
```
coverage(6, "NNK", N = 10^8)

## [1] 0.5229
```

It may be of interest to compare the expected coverage across the different encoding schemes available in **peptider**. We can plot the expected coverage as a function of the library size for all four built-in encoding schemes as follows:

```
library(plyr)
library(ggplot2)
```

```
dframe <- expand.grid(Scheme = c("NNN", "NNB", "NNK", "trimer"), Size = 10^(4:10))
results.dframe <- ddply(dframe, .(Scheme, Size), function(row) {
    coverage(6, as.character(row$Scheme), row$Size)
})
names(results.dframe)[3] <- "Coverage"
```

```
qplot(Size, Coverage, data = results.dframe, geom = "line", colour = Scheme,
    size = I(2)) + scale_x_log10(breaks = 10^(4:10)) + theme(legend.position = "bottom")
```



Relative Efficiency

In general, it is desireable to achieve a coverage as close as possible to one if we'd like the library to contain all possible peptides. One can achieve this with arbitrarily high probability by increasing the library size (N). However, doing so can drastically increase the cost of the library and may not always be possible. Ideally, the library should contain as high as possible diversity with as small as possible size. Relative Efficiency is a measure to quantify this, and is defined as:

$$R(N,k) = D(N,k)/N$$

As N increases, the relative efficiency subsequently decreases. An ideal peptide library from a cost-benefit perspective would contain both a high diversity (and hence high expected coverage) and still a high relative effeciency. Efficiency can be computed with **peptider** in the same way as coverage.

```
efficiency(6, "NNK", N = 10^8)

## [1] 0.3347
```

Peptide Coverage

The measures of library diversity included in **peptider** introduced to this point are useful to broadly assess the library performance. However, applications of peptide libraries often require knowledge

of the probability of observing particular peptides in the library. **peptider** helps to analyze the the probability of observing both the peptide of interest to the user, as well as peptides that are most similar.

Peptide Inclusion

The peptide inclusion probability is the probability of obtaining at least one instance of a particular peptide in the library. If X is defined as a random variable representing the number of occurrences of this peptide, the probability is

$$P(X \geq 1) = 1 - P(X = 0) \approx 1 - e^{-N \sum_i p_i}$$

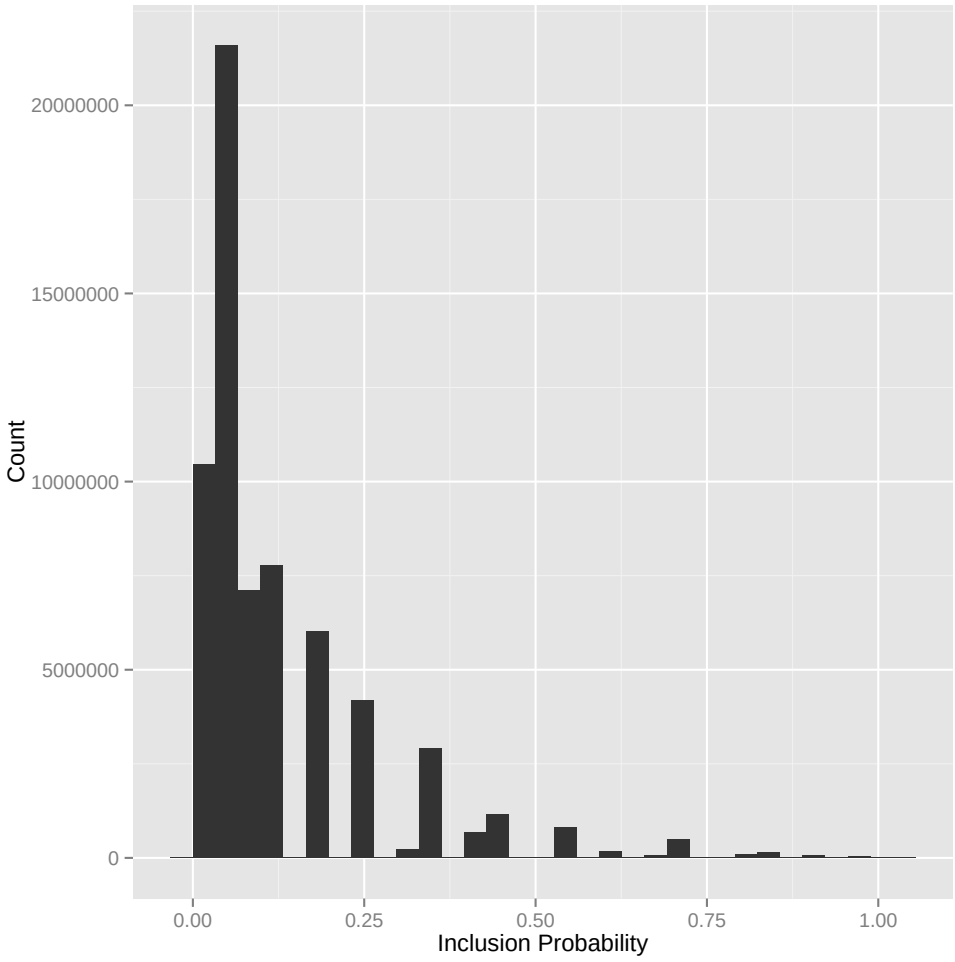
p_i again is the probability of peptide class i. Because of the dependence on the peptide class, this inclusion probability will vary depending on the particular peptide of interest, and the library scheme. The `ppeptide` function in **peptider** allows computation of the probability. It accepts a peptide sequence as a character vector, the library scheme, and the library size.

```
ppeptide("HENNING", "NNK", N = 10^10)

## [1] 0.5166
```

We may also be interested in the range of inclusion probabilities for all peptides rather than just a particular sequence. In this case, the `detect` function is useful. detect differs a bit syntactically as it requires the created library to be passed in as an argument. For an NNK library with peptide lengths 6 and library size 10^7, we have:

```
my_lib <- libscheme("NNK", 6)

summary(detect(my_lib, 10^7))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0112  0.1840  0.3330  0.3980  0.5560  1.0000
```

```
qplot(detect(my_lib, 10^7), weight = di, geom = "histogram", data = my_lib$data) +
    xlab("Inclusion Probability") + ylab("Count")

## stat_bin:  binwidth defaulted to range/30.  Use 'binwidth = x' to adjust this.
```



Neighborhoods

A related concern to inclusion of a particular peptide is inclusion of a peptide's "neighbors". A degree-n neighbor of a peptide p is any peptide that differs by at most n amino acids from p. For example, the length 7 peptide HENNING includes degree-1 neighbor QENNING as well as HQNNING, and a degree-2 neighbor YENNLNG. **peptider** includes a function for working with neighborhoods, `getNeighbors`, which accepts a peptide sequence as a character vector and returns all degree-1 neighbors.

```
getNeighbors("HENNING")

##  [1] "HENNING" "QENNING" "YENNING" "HDNNING" "HQNNING" "HKNNING" "HEDNING"
##  [8] "HENDING" "HENNLNG" "HENNMNG" "HENNVNG" "HENNIDG"
```

Although there is no built-in function for computing degree-2 and greater neighborhoods for a given peptide, the functionality can be emulated by successive calling of getNeighbors, i.e.,

```
unique(unlist(getNeighbors(getNeighbors("HENNING"))))

##  [1] "HENNING" "QENNING" "YENNING" "HDNNING" "HQNNING" "HKNNING" "HEDNING"
##  [8] "HENDING" "HENNLNG" "HENNMNG" "HENNVNG" "HENNIDG" "RENNING" "EENNING"
## [15] "KENNING" "QDNNING" "QQNNING" "QKNNING" "QEDNING" "QENDING" "QENNLNG"
## [22] "QENNMNG" "QENNVNG" "QENNIDG" "FENNING" "WENNING" "YDNNING" "YQNNING"
## [29] "YKNNING" "YEDNING" "YENDING" "YENNLNG" "YENNMNG" "YENNVNG" "YENNIDG"
## [36] "HNNNING" "HDDNING" "HDNDING" "HDNNLNG" "HDNNMNG" "HDNNVNG" "HDNNIDG"
## [43] "HRNNING" "HHNNING" "HQDNING" "HQNDING" "HQNNLNG" "HQNNMNG" "HQNNVNG"
```

```
## [50] "HQNNIDG" "HKDNING" "HKNDING" "HKNNLNG" "HKNNMNG" "HKNNVNG" "HKNNIDG"
## [57] "HEENING" "HEDDING" "HEDNLNG" "HEDNMNG" "HEDNVNG" "HEDNIDG" "HENEING"
## [64] "HENDLNG" "HENDMNG" "HENDVNG" "HENDIDG" "HENNLDG" "HENNMDG" "HENNVDG"
## [71] "HENNIEG"
```

Further Work

Although these functions are suitable for working with peptides up to about length ten, they become slower and more problematic for larger peptides. Work has progressed on a new suite of functions which simplify the encoding of peptides. By recognizing the assumption of independence of each amino acid, we can store counts of each peptide class in order to avoid storing all possible permutations of peptide classes. This greatly simplifies the computation time and allows for peptides of length 20 and beyond to be analyzed in the context of peptide libraries.

Replacement functions for most of the previously described functionality using this new encoding scheme is available unexported in peptider. For instance, to compute the coverage of a size 10^{25} NNK library with peptides of length 18, one can call:

```
peptider:::coverage_new(18, "NNK", N = 10^25)

## [1] 0.7923
```

Functions for coverage, efficiency, diversity, and inclusion probabilities have all been written and have the “new" suffix.

Conclusion

peptider aims to provide a seamless way to assess the quality of different peptide libraries. By providing functions to calculate both the diversity of the peptide library itself, as well as inclusion probabilities of particular peptides of interest, it can help researchers in other fields to make a more informed decision regarding which libraries to invest in.

Bibliography

M. B. Irving, O. Pan, and J. K. Scott. Random-peptide libraries and antigen-fragment libraries for epitope mapping and the development of vaccines and diagnostics. *Current opinion in chemical biology*, 5(3):314–324, 2001. [p1]

D. J. Rodi, R. W. Janes, H. J. Sanganee, R. A. Holton, B. Wallace, and L. Makowski. Screening of a library of phage-displayed peptides identifies human bcl-2 as a taxol-binding protein. *Journal of Molecular Biology*, 285(1):197 – 203, 1999. ISSN 0022-2836. doi: 10.1006/jmbi.1998.2303. URL http://www.sciencedirect.com/science/article/pii/S0022283698923038. [p1]

A Web Application for Efficient Analysis of Peptide Libraries

Eric Riemer Hare

January 25, 2014

Abstract

This paper introduces a web application called PeLiCa, or Peptide Library Calculator. Built upon the Shiny framework, PeLiCa provides an easy-to-use and powerful front-end to the R package *peptider*. PeLiCa allows users to conduct a statistical analysis of a set of pre-defined peptide library schemes, or a custom-defined scheme of their own choosing. Results are instantly displayed, allowing the user to make a more informed decision regarding what specific peptide library will be most useful for their particular application or research.

1 Introduction

With the introduction of the R package *peptider*, analysis of the statistical properties of various peptide libraries is now possible. However, use of this package still requires familiarity with the R language, and more general programming concepts. In this paper, I introduce a new web interface called *PeLiCa* (or Peptide Library Calculator) which provides a useable and flexible front-end to peptider. PeLiCa is designed for biologists and others working with peptide libraries, and does not require programming knowledge to conduct an analysis.

2 Structure

PeLiCa is a Shiny application[REF]. Shiny is a framework for writing web-applications in the R language, requiring little-to-no javascript programming knowledge. PeLiCa uses this framework to provide interactivity. For instance, when the user of PeLiCa changes a property of the peptide library, such as the encoding scheme, the results, tables, and plots will instantly update to reflect the new library. PeLiCa is currently hosted on the Glimmer server provided by the RStudio team.

3 User Interface

Similar to other Shiny applications, PeLiCa consists of three primary UI components, the Configuration Panel, the Tab Panel, and the Results Panel, each illustrated in Figure 1. The Configuration Panel is located along the left-hand column. This panel allows for various parameters of peptide libraries to be adjusted, and some configuration options relating to PeLiCa to be changed depending on user preference. The top panel is the Tab Panel, which contains various tabs corresponding to different properties of peptide libraries that can be investigated. The bottom panel is the Results panel, which will contain the results of the analysis depending on the tab and configuration options selected.

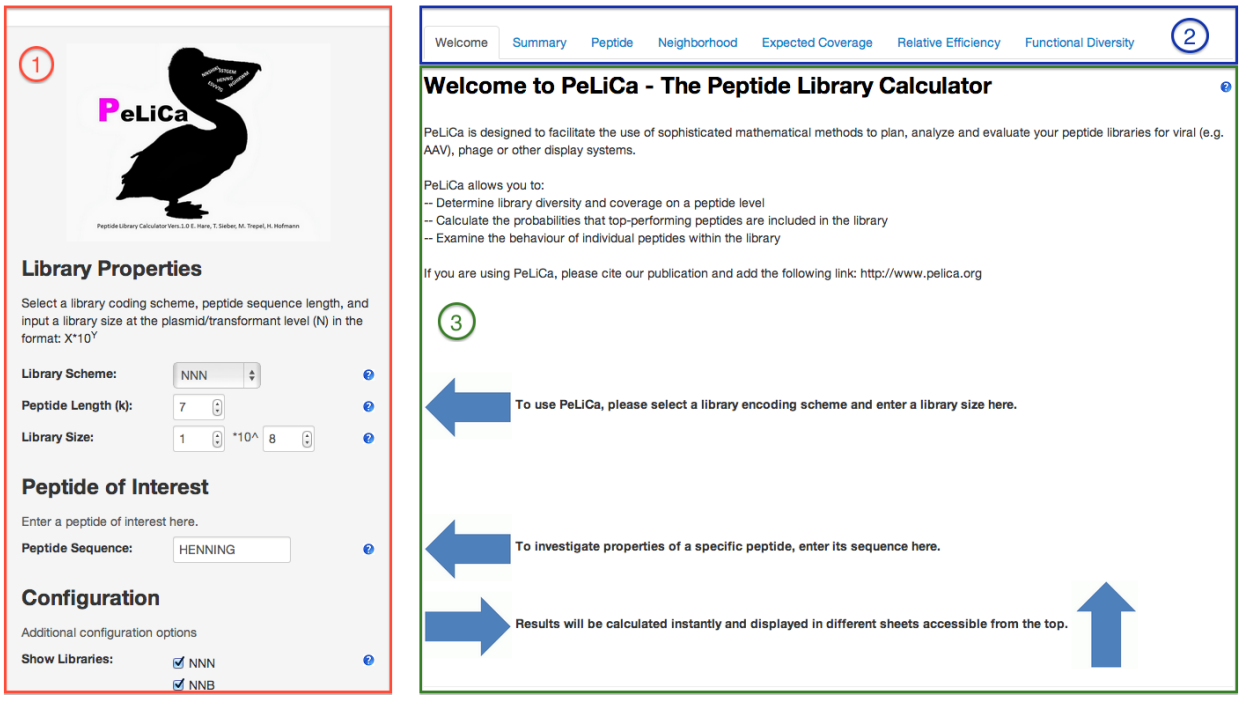


Figure 1: Screenshot of PeLiCa indicating the three primary UI components. (1) The Configuration Panel (2) The Tab Panel (3) The Results Panel.

Another important component of the user interface are the help tooltips. Throughout the application, blue question marks will be available. When the user moves their mouse cursor over these icons, helpful

tooltips will appear to instruct the user on how to proceed, or provide more information about the library property being investigated.

4 Features

4.1 Configuration Panel

The first set of features available in PeLiCa involve the specification of properties of the library of interest. These features are displayed in Figure 2. Users can first select a library scheme. PeLiCa provides several built-in library schemes, the first of which is the NNN scheme, in which all four bases (Adenine, Guanine, Cytosine, and Thymine) can occur at all three positions in a particular codon. The second is the NNB scheme, where the first two positions are unrestricted, but the third position can only be three bases. The third is NNK/S, which covers both NNK and NNS schemes, with the third position restricted to two bases. Both NNK and NNS have identical statistical properties in the analysis available in PeLiCa[REF]. Finally, there are trimer-based libraries in which the codons are pre-defined. PeLiCa also includes variations of these four library types in which Cysteine is treated as a non-viable amino acid.

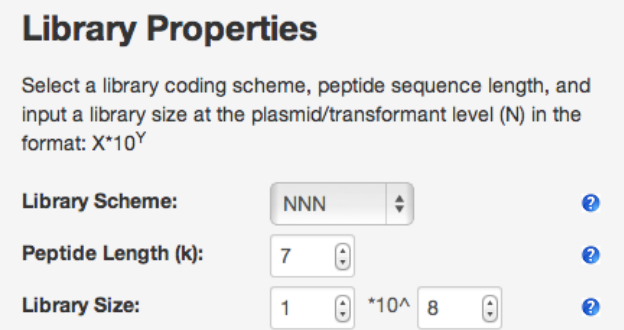


Figure 2: The Library Properties section of the Configuration Panel.

PeLiCa also has support for user-defined library schemes. If the user selects "Custom Scheme" for the library scheme, they will be presented with an upload dialog, along with a set of instructions for uploading a custom scheme. Using a custom scheme with PeLiCa requires a minimal use of programming and may be less suitable for those unfamiliar with R.

Users can also specify the peptide length and the library size. Peptide lengths can range from six to ten amino acids, with support for larger peptides coming soon. The library size is specified in scientific notation of the form $x \times 10^y$. The user will specify values for x and y in this equation. x can range from 1.0 to 9.9 in increments of 0.1, and y can range from six to 14 in increments of one, yielding a supported range of library sizes between 1.0×10^6 and 9.9×10^{14}. Support for large library sizes is also in progress.

Users can then specify a peptide of interest, as shown in Figure 3, and configure which other libraries to display in the Results Panel in Figure 4.

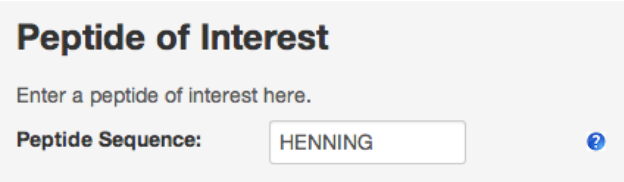


Figure 3: The Peptide of Interest section of the Configuration Panel.

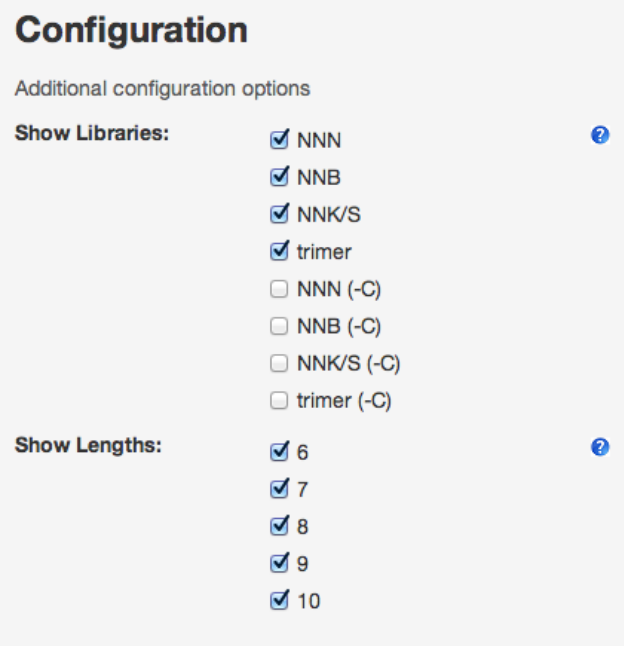


Figure 4: The Configuration section of the Configuration Panel.

4.2 Results Panel

The primary functionality for PeLiCa is available in the Results Panel, which are accessible through each of the tabs in the Tab Panel.

4.2.1 Welcome

PeLiCa begins on the Welcome tab. The Welcome tab provides information on the functionality of PeLiCa, and a quick guide for its use. The tooltip on this tab illustrates the system requirements.

4.2.2 Summary

The Summary tab contains most of the key information from the other tabs, condensed into an easy-to-digest format. First, information on your library is displayed. Some of this information includes the coverage, the peptide diversity, and the probabilities of peptide inclusion. Information on your selected peptide is displayed below this, summarizing the inclusion probability and the number of different DNA encodings of this particular peptide. Finally, a table displaying a randomly generated sample of peptides is shown at the bottom. This table includes the amino acids comprosing the peptide, the peptide class under the chosen encoding scheme, the number of DNA encodings, and the probability of inclusion in your library.

4.2.3 Peptide

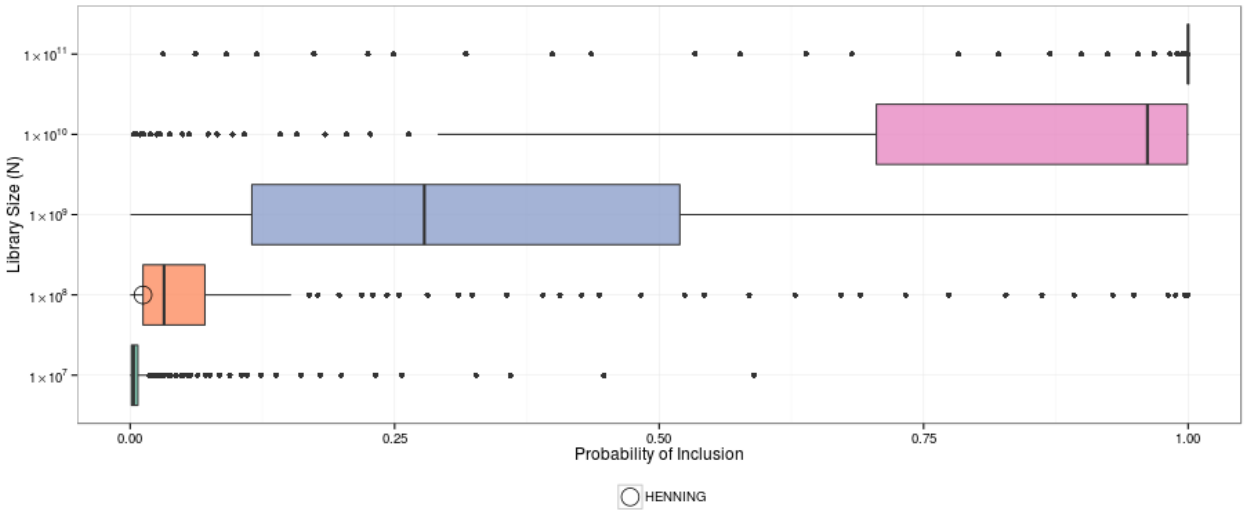


Figure 5: Boxplots displayed in PeLiCa representing the inclusion probabilities for an NNN library with peptide length seven and a library size of 1×10^8. The inclusion probability of HENNING is displayed in the plot as a circle.

4.2.4 Neighborhood

4.2.5 Expected Coverage

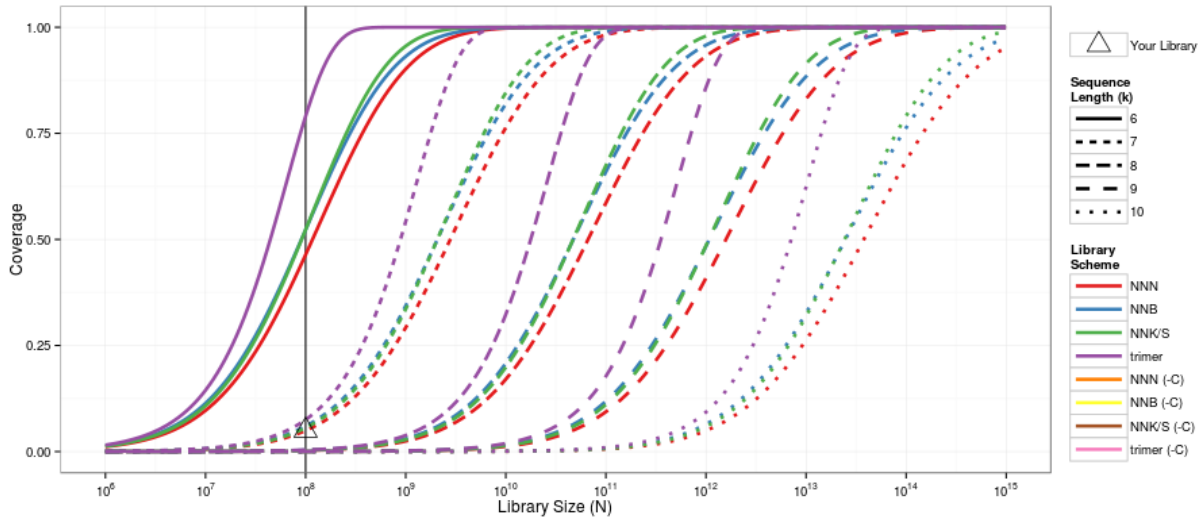


Figure 6: Plots of expected coverage for an NNN library with peptide length seven and a library size of 1×10^8

4.2.6 Relative Efficiency

4.2.7 Functional Diversity

5 Further Work

A new version of PeLiCa is in currently in progress. The new version supports lower resolution monitors, includes a new framework for tooltips, and supports a wider range of peptide lengths and library sizes. This new version is currently deployed on ShinyApps at `http://erichare.shinyapps.io/pelica`.

6 Conclusion