An Active Learning Approach to Finding Minimally-sized Peptide Substrates

Michael Burkhart* Peter I. Frazier† Nathan Gianneschi*
Michael K. Gilson‡

(author order currently alphabetical, other authors to be added as we go)
November 15, 2013

1 Problem Description

We have two enzymes (Sfp from *Bacillus subtilis*, and PaAcpH from *Pseudomonas aeruginosa*, and a collection of peptides that can potentially act as a substrate for one or both of these enzymes. Our goal is to find a peptide that acts as a substrate for both of these enzymes, and is as short as possible.

To support this goal, we can do lab experiments, in which we synthesize a peptide and test, for each enzyme, whether it is a substrate.

We also have an initial dataset gathered from the literature, which contains both peptides from nature, and peptides discovered using phage display. Each peptide in this dataset is labeled as to whether or not it is a substrate for enzyme 1 (Sfp), and similarly for enzyme 2 (PaAcpH). All peptides in the dataset are substrates for enzyme 1, and 8 out of 15 of them are substrates for enzyme 2.

We are designing an algorithm that suggests which peptide to synthesize and test next, so as to reach our goal with as few experiments as possible.

Experiments are done one peptide at a time, and so the algorithm suggests only one peptide at a time, waiting for the results from the experiment before suggesting the next peptide. A large collection of peptides would be considered by the algorithm for potential synthesis and testing, e.g., all peptides with length less than a given threshold. That is, we would consider more peptides than just those that are sub-peptides of peptides from the literature known to be substrates for one enzyme.

Here is a high level view of how work would flow:

1. Using the data from the literature, we train a classifier, whose output is not just an estimate of whether or not the peptide will be a substrate for each enzyme, but also

*Department of Chemistry and Biochemistry, UC San Diego
†School of Operations Research & Information Engineering, Cornell University
‡Skaggs School of Pharmacy and Pharmaceutical Sciences, UC San Diego

the probability of each possibility. This document proposes using Naive Bayes as our classification method, but other classification methods are also possible. (Section 2)

2. While a short-enough peptide acting as a substrate for both enzymes has not yet been found, and we have the budget for additional experiments, we would do the following:
   (a) Based on the most recent Bayesian prior/posterior distribution, use a mathematical method described below to suggest which peptide to synthesize and test next. (Section 3)
   (b) Synthesize and test this peptide in the lab.
   (c) Update the classifier to incorporate the most recent experimental results. (Section 2)

2 Classification Method

As an initial method, we propose using Naive Bayes, which actually performs quite well in many applications, despite the name. This is a standard method, but we briefly review it in the context of this problem, and to introduce notation that is used later.

We represent a peptide by its sequence of amino acids, and refer to the index of an amino acid based on its position relative to the serine. The serine itself is at position 0; the amino acid immediately next to the serine, toward the N-terminal, is at position -1; one amino acid closer to the N-terminal is -2, etc. Similarly, the amino acid next to the serine, toward the C-terminal, is at position 1, then next to that is position 2, etc. We will consider positions from -20 up to 20.

We additionally propose using a reduced amino acid alphabet, and have begun work using the Dayhoff classes described in [**?**]. There are 6 Dayhoff classes. For a given peptide, we let A_i be the amino acid at position i, and we let X_i be the Dayhoff class of this amino acid. We refer to a full peptide as X or x (we use a capital letter if we are thinking of the peptide as being drawn at random)[1]. For each enzyme $k = 1, 2$, let $Y(x, k)$ be 1 if peptide x is a substrate for enzyme k, and 0 if not.[2]

The Naive Bayes classifier supposes that, if we fix one of the enzymes $k = 1, 2$, and we draw a peptide X at random from the set of peptides for which $Y(X, k) = 1$, i.e., from the set of peptides that act as a substrate for enzyme k, then for each $i \neq 0$,

$$P\left(X_i = j | Y(X,k) = 1, \theta^{(k)}\right) = \theta^{(k)}_{1,i}(j) \tag{1}$$

where $j \in \{1, \ldots, 6\}$ represents one of the six Dayhoff classes, and the values $\theta^{(k)}_{1,i}(j)$, with $j = 1, \ldots, 6$ and $i = -20, \ldots, -1, 1, \ldots, 20$ are parameters to be estimated.[3] Moreover, we

[1] Note that this is potentially ambiguous, since two amino acids with the same sequence of Dayhoff classes would be referred to with the same value of X. We may need to modify this notation, but we ignore this ambiguity for now.

[2] My understanding is that if $Y(x, 1) = 0$, then it is not possible to test whether $Y(x, 2)$ is 0 or 1. In this case, we assume that $Y(x, 2) = 0$. Although this implies that $Y(x, 1)$ is not independent of $Y(x, 2)$, we will nevertheless assume it is independent, at least in the first version of this approach, for simplicity.

[3] Since all peptides considered will have a serene, its presence does not allow us to differentiate substrates from non-substrates, so we simply skip $i = 0$.

assume that, conditioned on $Y(X,k)=1$, $\theta^{(k)}$, X_i is independent across i. This assumption is where the "naive" comes from in Naive Bayes.

We use a similar model for peptides that are not a substrate for enzyme k. That is, if we draw a peptide X at random from the set of peptides for which $Y(X,k)=0$, then for $i \neq 0$,

$$P\left(X_i = j | Y(X,k)=0, \theta^{(k)}\right) = \theta^{(k)}_{0,i}(j) \tag{2}$$

for $j \in \{1,\ldots,6\}$. Here as before $\theta^{(k)}_{0,i}(j)$, are parameters to be estimated.

We additionally assume some known prior distribution $P(Y(x,k)=1)$ for each $k=1,2$. Initially, we choose $P(Y(x,k)=1) = P(Y(x,k)=0) = 1/2$.

Let $\theta^{(k)}$ represent the full set of parameters, $\theta^{(k)}_{y,i}(j)$, for $y=0,1$, $j=1,\ldots,6$ and $i=-20,\ldots,-1,1,\ldots,20$. If we know $\theta^{(k)}$, then we can calculate the probability that $Y(x,k)=1$, i.e., the probability that peptide x is a substrate for enzyme k, using Bayes rule. This quantity is

$$P\left(Y(x,k)=1|\theta^{(k)}\right) = \frac{P(Y(x,k)=1)\prod_i \theta^{(k)}_{1,i}(x_i)}{\left[P(Y(x,k)=1)\prod_i \theta^{(k)}_{1,i}(x_i)\right] + \left[P(Y(x,k)=0)\prod_i \theta^{(k)}_{0,i}(x_i)\right]} \tag{3}$$

where the product over i is taken over all amino acids in the peptide, except for the serine at position 0, and $P(Y(x,k)=1)$ (respectively $P(Y(x,k)=0)$) is the probability that a randomly chosen peptide will be a substrate for enzyme k (respectively, not a substrate).

If we choose $P(Y(x,k)=1) = P(Y(x,k)=0) = 1/2$, then these terms cancel, to obtain

$$P\left(Y(x,k)=1|\theta^{(k)}\right) = \frac{\prod_i \theta^{(k)}_{1,i}(x_i)}{\left[\prod_i \theta^{(k)}_{0,i}(x_i)\right] + \left[\prod_i \theta^{(k)}_{1,i}(x_i)\right]} \tag{4}$$

Estimation of θ: We can estimate the $\theta^{(k)}_{y,i}(j)$ values using Bayesian inference. We begin with a prior probability distribution on each vector $\theta^{(k)}_{y,i}$, which is Dirichlet$(\alpha^{(k)}_{y,i}(1),\ldots,\alpha^{(k)}_{y,i}(6))$, where $\alpha^{(k)}_{y,i} = (\alpha^{(k)}_{y,i}(1),\ldots,\alpha^{(k)}_{y,i}(6))$ parameterizes the prior distribution. A good initial choice for this parameter vector seems to be to choose $\alpha^{(k)}_{y,i}(j)$ to be constant across j, k and y, and to only depend upon i. We choose this value to be 1 when $i=\pm 1$, and to increase as i moves further from 0, which says that amino acids further from the serine are less likely to have a strong influence on activity.

With this prior distribution, our posterior distribution on $\theta^{(k)}_{y,i}$ is also a Dirichlet distribution, but with different parameters. In particular, it is Dirichlet$(\alpha^{(k)}_{y,i}(1)+N^{(k)}_{y,i}(1),,\ldots,\alpha^{(k)}_{y,i}(6)+N^{(k)}_{y,i}(6))$, where $N^{(k)}_{y,i}(j)$ counts how many peptides x in the training data with $Y(x,k)=y$ had $x_i=j$. That is, it counts how many peptides had amino acid i in Dayhoff class j. The posterior remains independent across y,i,k. The expectation of $\theta^{(k)}_{y,i}(j)$ under this posterior distribution is

$$\theta^{(k)}_{y,i}(j) \approx \frac{\alpha^{(k)}_{y,i}(j) + N^{(k)}_{y,i}(j)}{\sum_{j'} \alpha^{(k)}_{y,i}(j') + N^{(k)}_{y,i}(j')}. \tag{5}$$

Computation of $P(Y(x,k)=1|$training data$)$ To calculate the probability that $Y(x,k) = 1$ given the training data, we must integrate (4) over the full posterior distribution $\theta_{y,i}^{(k)}$. A simpler approximate method for calculating $P(Y(x,k)=1|\text{training data})$ is to plug in the estimates of $\theta_{y,i}^{(k)}$ from (5) into (4).

3 Method for Suggesting Which Peptide to Test Next

One approach to recommending peptides to test, based on estimates obtained from a statistical model, would be to assume that the estimates of substrate / not substrate reported by the statistical model were correct, and then find the shortest peptide that the statistical model estimated to be a substrate for both enzymes. However, this approach ignores the fact that predictions from a statistical model are only correct with some probability.

The following method is more sophisticated, and incorporates the probability of correctness into what it recommends testing. If the probabilities of substrate / not substrate are all 0 or 1, then we recover the less sophisticated approach. We use the probabilities of substrate / not substrate as computed by the statistical method described in Section 2, but the same method could be equally well applied with probabilities estimated in another way.

We first describe a method for computing the value $V(x)$ of synthesizing and testing any given peptide x. Then, the method for recommending which peptide to test next is then to recommend the peptide for which $V(x)$ is the largest.

The method for computing $V(x)$ depends on L^*, which is the length of the smallest peptide found so far that is an enzyme for both substrates. Let $L(x)$ be the length of peptide x. Let

$$p(x) = P\left(Y(x,1) = Y(x,2) = 1|\text{training data}\right).$$

Under the model described in Section 2, with $Y(x,1)$ independent of $Y(x,2)$, this is computed as $p(x) = P\left(Y(x,1)=1|\text{training data}\right) P\left(Y(x,2)=1|\text{training data}\right)$, where each $P(Y(x,k) = 1|\text{training data})$ can be computed using either the exact or the approximate method described at the end of Section 2.

If we test peptide x, and $Y(x,1) = Y(x,2) = 1$, then the new value of L^* will be $\min(L^*, L(x))$, and the reduction in L^* will be $\max(0, L^* - L(x))$. If, instead, at least one of $Y(x,1)$ or $Y(x,2)$ is different from 1, then L^* remains unchanged, and the reduction in L^* is 0. We let $V(x)$ be the expected value of this reduction in length, due to testing peptide x,

$$V(x) = p(x)\max(0, L^* - L(x)).$$

Then, the peptide that we suggest testing next is

$$\arg\max_x V(x)$$

This is similar to expected improvement methods, which are used in the optimization literature [**?**], and to knowledge-gradient method [**?**, **?**]).

Computation of expected improvement of a set of peptides Given a set of N peptides: $S = (X_n)_{n=1}^N$, we want to compute a value of this set. Suppose the shortest peptide that is substrate for both enzymes in S is x', and $L(x')$ is the length of x', then the value of S can be defined as:

$$V(S) = \max(0, L^* - L_{x'}).$$

For simplicity, we assume the peptides in S are sorted in the order of increasing length. $\forall n \in \{1, ..., N\}$, let $p_{min}(X_n)$ denote the probability that X_n is the shortest peptide that is substrate for both enzymes given training data. Let

$$Y(x) = Y(x, 1) \cdot Y(x, 2)$$

Then $Y(x) = 1$ if and only if x is substrate for both enzymes, and

$$p_{min}(X_n) = P(Y(X_n) = 1, Y(X_{n+1}) = ... = Y(X_N) = 0|\text{training data})$$

And the value of S can be represented as:

$$V(S) = \sum_{x' \in S} p_{min}(x') \max(0, L^* - L(x'))$$

We call $V(S)$ the expected improvement of S.

Since $Y(x, k) = 1|$training data and $Y(x', k) = 1|$training data are not independent, we don't have a closed form of $P_{min}(x)$. Therefore we compute $V(S)$ through simulation.

We carry out M iterations of simulation. In each run, we sample θ from posterior distribution given training data. We then use the sampled θ to predict $P(Y(X_n, 1) = 1, Y(X_n, 2)) = 1|$training data) and sample $Y(X_n, 1)$ and $Y(X_n, 2)$ from the prediction for all $n \in \{1, ..., n\}$. Then we choose the peptide x with shortest length and $Y(x, 1) = Y(x, 2) = 1$, and calculate the reduction in length compared with L^*. Finally we average the reduction calculated in each iteration to get expected improvement.

Require: inputs: M, training data, $S = (X_n)_{n=1}^N$ and L^*
$(P_n)_{n=1}^N \leftarrow 0$
$(I_n)_{n=1}^N \leftarrow 0$
for $n = 1$ to N **do**
  $R_n \leftarrow \max(0, L^* - L(X_n))$
end for
$(\text{maxR}_i)_{i=1}^M \leftarrow 0$
for $i = 1$ to M **do**
  $\theta \leftarrow$ sample from posterior distribution given training data
  $(P_n)_{n=1}^N \leftarrow$ predict from Naive Bayes classifier θ
  for $n = 1$ to N **do**
    $I_n \leftarrow$ sample from Bernoulli(P_n)
  end for
  $\text{maxR}_i \leftarrow \max_{1 \le n \le N} I_n \cdot R_n$
end for
$V(S) \leftarrow \frac{1}{N} \sum_{i=1}^N \text{maxR}_i$

Knowledge Gradient method We introduce a knowledge-gradient approach to find a set of peptides with a large expected improvement. The problem is as follows: given a set of N randomly chosen a peptides $S = (X_n)_{n=1}^N$, we want to modify some peptides in S so that the expected improvement $V(S)$ can be as large as possible. To achieve this goal, we iteratively carry out the following process: we first randomly choose a peptide in S and a position in that peptide, and we use a knowledge gradient policy to decide which peptide class in that position will maximize the expected improvement of S. This policy works as follows.

Let C be the number of classes in our reduced alphabet of amino acids. We have C alternative classes for the selected position in the selected peptide. Let $\mathcal{C} = \{1, ..., C\}$ be our set of choices and $c \in \mathcal{C}$ represent an alternative. Let $V_c(S)$ denote the expected improvement given c is the amino acid class at the chosen position. We assume $V_c(S) \sim \mathcal{N}(\mu_c, \sigma_c^2)$.

Given our measuring budget K(the maximum number of measurements), in each measurement, we choose an alternative $c \in \{1, ..., C\}$ and simulate the desired expected improvement once and update our belief about $V_c(S)$. After K measurements, we simply pick the class with highest μ to be the class at the chosen position. The key problem is which alternative we choose to measure at each iteration.

After k iterations, we let μ^k be our vector of means and β^k vector of precisions(inverse of variances). We compute a knowledge gradient factor $\nu_c^{KG,k}$ of each alternative c. Let c^{k+1} denote the alternative we choose to measure in the $(k+1)$th iteration. The knowledge gradient policy assigns

$$c^{k+1} = \arg\max_{c \in \mathcal{C}} \nu_c^{KG,k}$$

After simulating the expected improvement of S: W^{k+1}, assuming c^{k+1} is the class at the chosen position, we update our belief about $V(S)$ using the following equations:

$$\begin{aligned}
\mu_c^{k+1} &= \begin{cases} \frac{\beta_c^k \mu_c^k + \beta_c^W W^{k+1}}{\beta_c^k + \beta_c^W} & \text{if } c^{k+1} = c \\ \mu_c^k & \text{otherwise} \end{cases} \\
\beta_c^{k+1} &= \begin{cases} \beta_c^k + \beta_c^W & \text{if } c^{k+1} = c \\ \beta_c^k & \text{otherwise} \end{cases}
\end{aligned}$$

Here β^W is the measurement precision. To obtain β_c^W for alternative c, we simply simulate the corresponding expected improvement for a small number of times and use the inverse of sample variance of our simulated data as the measurement error.

Below is the process of computing the knowledge gradient factor $\nu^{KG,k}$. First, we compute the the variance in our belief of μ^{k+1} given μ_k:

$$\begin{aligned}
\tilde{\sigma}_c^{2,k} &= Var[\mu_c^{k+1} | \mu^k, \beta^k] \\
&= Var[\mu_c^{k+1} - \mu_c^k | \mu^k, \beta^k]
\end{aligned}$$

It can be shown that

$$\begin{aligned}\tilde{\sigma}_c^{2,k} &= \sigma_c^{2,k} - \sigma_c^{2,k+1} \\ &= (\beta_c^k)^{-1} - (\beta_c^k + \beta_c^W)^{-1}\end{aligned}$$

.

The knowledge gradient is found by first computing

$$\zeta_c^k = -\left|\frac{\mu_c^k - \max_{c' \neq c} \mu_{c'}^k}{\tilde{\sigma}_c^k}\right|$$

which is called the normalized influence of decision c. We then compute

$$f(\zeta) = \zeta\Phi(\zeta) - \phi(\zeta)$$

where $\Phi(\zeta)$ and $\phi(\zeta)$ are cumulative standard normal distribution and normal density respectively. Finally, the knowledge gradient factor is computed as

$$\nu_c^{KG,k} = \tilde{\sigma}_c^k f(\zeta_c^k)$$

References