No title

Erik Hartman[2] Jonas Wallin[1]

[1] Department of Statistics, Lund University
[2] Division of Infection Medicine, Faculty of Medicine, Lund University

Algorithms

Algorithm 1: Simulate proteolysis of protein P

Input: protein P, $n_{generate}$ θ_{enzyme}, θ_{gamma}, p_{endo}, p_{exo}
Output: F

$f \leftarrow$ dict[sequence : count]
$f(P) \leftarrow n_P$
while $i < n_{generate}$ **do**
  $n_{generate} \leftarrow \sum_x T(x)$
  $x \sim U(0,1)$
  if $x > p_{endo}$ **then**
    //exo-protease
    sequence to chew $s \sim f$ //sample sequence from dict with weights count
    $x \sim U(0,1)$
    $a \leftarrow gamma(len(s), \theta_{gamma})$
    if $a < x$ **then**
      //accept
      $F(P) \leftarrow F(P) + 1$
      $n_{generate} \leftarrow n_{generate} + 1$
    end
  else
    //endo
    $f_{cut}(s) \leftarrow \sum_s N^s_{aa} * \theta_{aa}$
    sequence to cut $s \leftarrow f_{cut}(s)$//sequence to cut
    first index to cut $index_a \sim s(\theta_{aa}(aa_x))$
    $index_2 \sim s(\theta_{aa}(aa_x)) * gamma(|index_1 - index_2|)$
    left $\leftarrow s[: min(index_1, index_2) + 1]$
    middle $\leftarrow s[min(index_1, index_2) + 1 : max(index_1, index_2)]$
    right $\leftarrow s[max(index_1, index_2) + 1 :]$
    if *len(middle)>5* **then**
      $f(middle) \leftarrow f(middle) + 1$
      $n_{generate} \leftarrow n_{generate} + 1$
    end
    for *s in [left, right]* **do**
      $x \sim U(0,1)$
      $a \leftarrow gamma(len(s), \theta_{gamma})$
      if $a < x$ *and s is not terminal peptide in* P **then**
        //accept
        $F(P) \leftarrow F(P) + 1$
        $n_{generate} \leftarrow n_{generate} + 1$
      end
    end
  end
end
Returns f

Algorithm 2: Estimating θ numerically. To generate a guess, simulate degradation of protein P with parameters θ to generate $n_{generate}$ peptides.;

Input: protein P, n_P, true distribution T, θ, lr_{endo}, lr_{exo}
Output: θ

for *i from 0 to* n_{endo} **do**
  Generate guess G;
  Compute loss $L \leftarrow D_{KL}(G||T) + D_{KL}(T||G)$;
  for *each amino acid aa in* θ_{aa} **do**
    $\theta_{aa}(aa) \leftarrow \theta_{aa}(aa) + lr_{endo}$;
    Generate guess $\hat{G}$ with new θ;
    Compute new loss $\hat{L} \leftarrow D_{KL}(\hat{G}||T) + D_{KL}(T||\hat{G})$;
    while $\hat{L} < L$ **do**
      Compute weighted learning rate $lr_w \leftarrow lr_{endo} * \hat{L} - L$;
      $L \leftarrow \hat{L}$;
      $\theta_{aa}(aa) \leftarrow \theta_{aa}(aa) + lr_w$;
      Generate guess $\hat{G}$ with new θ;
      Compute new loss $\hat{L} \leftarrow D_{KL}(\hat{G}||T) + D_{KL}(T||\hat{G})$;
    end
    $\theta_{aa}(aa) \leftarrow \theta_{aa}(aa) - lr_{endo}$ //revert the initial parameter-change (before while-loop);
  end
end
Generate guess G;
Compute loss $L \leftarrow D_{KL}(G||T) + D_{KL}(T||G)$;
for *i from 0 to* n_{exo} **do**
  $x \leftarrow$ uniformly random from $1, -1$;
  $e \leftarrow lr_{exo} * x$;
  $\theta_{exo} \leftarrow \theta_{exo} + e$;
  Generate guess $\hat{G}$ with new θ;
  Compute new loss $\hat{L} \leftarrow D_{KL}(\hat{G}||T) + D_{KL}(T||\hat{G})$;
  if $\hat{L} > L$ **then**
    $\theta_{exo} \leftarrow \theta_{exo} - e$;
  else
    $L \leftarrow \hat{L}$;
  end
end
Return θ;