STA 206 Final Project : Novozymes Enzyme Stability Prediction

Greg DePaul

Project Description

Overview

The company Novozymes has released a Kaggle competition asking competitors to be able to predict enzyme thermostability. This work is valuable because the thermal stability of protein determines how much they can perform under harsh application conditions and/or their efficiency in serving as catalysts. This work is so valuable in fact that the top 3 scores win a cash prize, first place consisting of $12,000!

Unprobed Dataset Variables

```
training_data <- read.csv(file = 'train.csv')
keeps <- c("protein_sequence", "pH","tm")
training_data <- training_data[keeps]

update_training_data <- read.csv(file = 'train_updates_20220929.csv')
for (row in 1:nrow(update_training_data)) {
  if(update_training_data[row, "protein_sequence"] != "") {
    training_data[row, "protein_sequence"] <- update_training_data[row, "protein_sequence"]
    training_data[row, "pH"] <- update_training_data[row, "pH"]
    training_data[row, "tm"] <- update_training_data[row, "tm"]
  }
}

sapply(training_data,class)
```

```
## protein_sequence               pH               tm
##      "character"        "numeric"        "numeric"
```

- **Protein Sequence:** A variable length sequence of 20 possible states. An example sequence:

  VPVNPEPDATSVENVALKTGSGDSQSDPIKADLEVKGQSALPFDVDCWAILCKGAPNVLQRVNEKTKNSNRDRSGANKGPFKDPQKWGIKALPPKNPSWSAQDFKS

- **pH:** The acidity level at which these sequences thermostability were measured.
- **tm:** The measured thermostability. Response Variable.

```
print(nrow(training_data))
```

```
## [1] 31390
```

```
N <- nrow(training_data)
testing_data <- read.csv(file = 'test.csv')
print(nrow(testing_data))
```

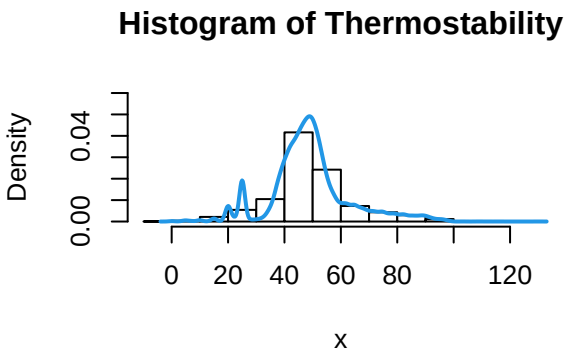
```
## [1] 2413
```

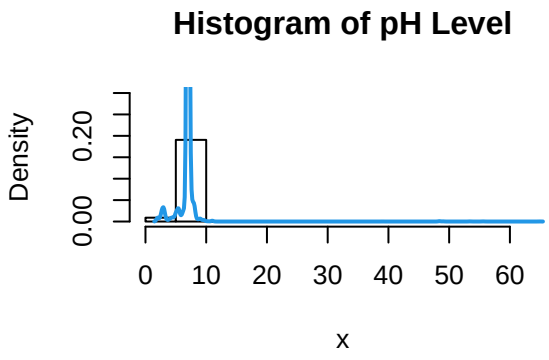
```
training_data$pH[is.na(training_data$pH)] <- 7
```

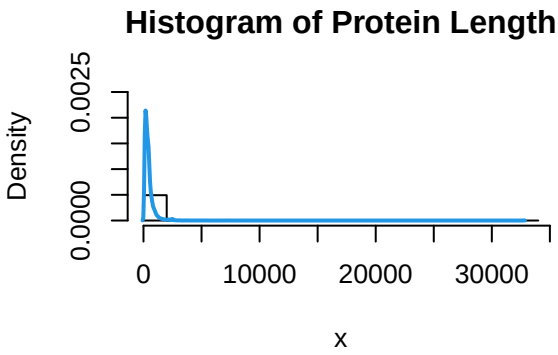
The number of given points in this dataset is 313901 There is also an competition set with 2413 sequences.

Initial Data Exploration

```
print_hist <- function(x, title, scale) {

  x2 <- seq(min(x), max(x), length = 40)

  fun <- dnorm(x2, mean = mean(x), sd = sd(x))

  hist(x, prob = TRUE, ylim = c(0, scale), col = "white", main=title)
  lines(density(x), col = 4, lwd = 2)
}

par(mfrow = c(2, 2))

print_hist(training_data$tm, 'Histogram of Thermostability',0.06)

print_hist(training_data$pH, 'Histogram of pH Level',0.3)

print_hist(nchar(training_data$protein_sequence), 'Histogram of Protein Length',0.0025)
```







Modeling

Make the assumption that this model is NOT language in nature.

Featurization

We can featurize this dataset by taking every sequence and embedding into a larger space defined by:

- **3 character sequence:** Break each sequence into its component 3 character sequence. Since each sequence represents the chaining of a chemical compound, we make the assumption that it is within this chaining that instability occurs. Since there are 20 achievable states, then this variable lies within a 8000 dimensional space.
- **Prevelance in dataset:** The number of times this particular sequence occurs within the greater dataset at large. Helps identify outliers / special sequences.
- **Expected value:** The empirical expected value for a sequence. Most sequences will average to the same mean value.
  - Max tm
  - Min tm
  - Mean Expected tm
  - Upper Quartile tm
  - Lower Quartile tm
- **Variance for Char Sequence:** How much variance does a character sequence experience. Therefore, common sequences should have less affect on the final outcome.

```
phrase_stats <- read.csv(file = 'phrase_stats.csv')
sapply(phrase_stats,class)
```

```
##                 X       subsequence              prev               std
##         "integer"       "character"         "integer"         "numeric"
##         median_ph           mean_tm            max_tm            min_tm
##         "numeric"         "numeric"         "numeric"         "numeric"
##         median_tm lower_quartile_tm upper_quartile_tm
##         "numeric"         "numeric"         "numeric"
```

```
prevelance_dictionary <- hash()
variance_tm_dictionary <- hash()
median_ph_dictionary <- hash()
mean_tm_dictionary <- hash()
max_tm_dictionary <- hash()
min_tm_dictionary <- hash()
median_tm_dictionary <- hash()
upper_quartile_tm_dictionary <- hash()
lower_quartile_tm_dictionary <- hash()

for (row in 1:nrow(phrase_stats)) {
  curr_phrase = phrase_stats[row, "subsequence"]
  prevelance_dictionary[[curr_phrase]] <- phrase_stats[row, "prev"]
  variance_tm_dictionary[[curr_phrase]] <- phrase_stats[row, "std"]
  median_ph_dictionary[[curr_phrase]] <- phrase_stats[row, "median_ph"]
  mean_tm_dictionary[[curr_phrase]] <- phrase_stats[row, "mean_tm"]
  max_tm_dictionary[[curr_phrase]] <- phrase_stats[row, "max_tm"]
  min_tm_dictionary[[curr_phrase]] <- phrase_stats[row, "min_tm"]
  median_tm_dictionary[[curr_phrase]] <- phrase_stats[row, "median_tm"]
  lower_quartile_tm_dictionary[[curr_phrase]] <- phrase_stats[row, "lower_quartile_tm"]
  upper_quartile_tm_dictionary[[curr_phrase]] <- phrase_stats[row, "upper_quartile_tm"]
}
```

Since the number of combinations is 8000, yet we only see 3910 states, we have significant data reduction!

Modeling

Given an enzyme sequence $(\{s_i\}_{i=1}^n, pH)$ pair, we consider important multivariable functions, similar to those we study in this class:

1. The **Predictive Value**, which should represent how much we trust an individual character sequence:

$$\text{predictive value}(s_i, pH) \sim \alpha_0 + \frac{\alpha_1}{\text{prevalence}(s_i)} + \frac{\alpha_2}{\text{variance}(s_i)} + \frac{\alpha_3}{1 + |pH - \text{median}\{pH(s_i)\}|}$$

2. The **Expected Value**, which represents what thermostability that sequence will likely take on based off that subset alone:

$$\text{expected value}(s_i) \sim \beta_0 + \beta_1 \text{mean}(s_i) + \beta_2 \max(s_i) + \beta_3 \min(s_i) + \beta_4 \text{median}(s_i) + \beta_5 \text{lower quartile}(s_i) + \beta_6 \text{upper quartile}(s_i)$$

We then construct a predictor model to be:

$$Y_{pred}(\{s_i\}_{i=1}^n, pH) = \frac{\sum_{i=1}^n \text{predictive value}(s_i, pH) \cdot \text{expected value}(s_i)}{\sum_{i=1}^n \text{predictive value}(s_i, pH)}$$

```
sigmoid <- function(z) {
  return(1/(1 + exp(-z)))
}

predictive_value <- function(s_i,curr_ph) {
  if(prevelance_dictionary[[s_i]] == 0) {
    print(s_i)
  }
  return(abs(1/prevelance_dictionary[[s_i]] + 1/(1 + variance_tm_dictionary[[s_i]]) + 1/(1 + abs(curr_ph - med
}

#predictive_value <- function(alpha,s_i,curr_ph) {
#  return(1 + abs(alpha[1] + alpha[2]*prevelance_dictionary[[s_i]] + alpha[3]*(1 + variance_tm_dictionary[[s_i
#}

expected_value <- function(beta,s_i) {
  return(beta[1] + beta[2]*mean_tm_dictionary[[s_i]] + beta[3]*max_tm_dictionary[[s_i]] + beta[4]*min_tm_dicti
}

predictor <- function(param_vec,curr_sequence,curr_ph) {
  num_val <- 0
  den_val <- 1

  for (k in 1:(nchar(curr_sequence)-2)) {
    curr_substr <- substr(curr_sequence, k, k+2)
    pred_val <- predictive_value(curr_substr,curr_ph)
    exp_val <- expected_value(param_vec,curr_substr)
    num_val <- num_val + pred_val*exp_val
    den_val <- den_val + pred_val
   }

   return(num_val / den_val)
}

# predictor <- function(param_vec,curr_sequence,curr_ph) {
#   num_val <- 0
#   den_val <- 1
#
#   for (k in 1:(nchar(curr_sequence)-2)) {
```

```
#     curr_substr <- substr(curr_sequence, k, k+2)
#     pred_val <- predictive_value(param_vec[1:4],curr_substr,curr_ph)
#     exp_val <- expected_value(param_vec[5:12],curr_substr)
#     num_val <- num_val + pred_val*exp_val
#     den_val <- den_val + pred_val
#    }
#
#    return(num_val / den_val)
# }
```

Loss Function Optimization

Our predictor function is nonlinear and therefore we turn to optimization to find our coefficient estimators. We define the loss function be

$$\mathcal{L}(Y_{pred}, Y_{true}) = \frac{1}{N}\sum_{k=1}^{N}(Y_{pred}^{(k)} - Y_{true}^{(k)})^2$$

where $N < 28,981$. We can then optimize by minimizing this loss function over the few linear coefficients chosen above in order to minimize loss.

```
# L <- function(gamma) {
#   sum <- 0
#   trainIndex <- sample(N, 1000)
#
#   for (row in trainIndex) {
#     curr_ph <- training_data[row, "pH"]
#     Y_pred = predictor(gamma,training_data[row, "protein_sequence"],curr_ph)
#     Y_true = training_data[row, "tm"]
#     sum <- sum + (Y_pred - Y_true)^2
#   }
#     return(sum/1000)
#   }
#
# minimum = c(-10,-10,-10,-10,-10,-10,-10)
# maximum = -1*minimum
# res <- optimx(par = c(1,1/6,1/6,1/6,1/6,1/6,1/6), fn = L, lower=minimum,upper=maximum,control=list(maxit=10))
# summary(res)
# min_par <- res$par

#print(L(c(0,0,0,0,0.5,0.5,0.5,0.5,0.5,0.5,0.5)))
```

```
  # min_par = c(1,1/6,1/6,1/6,1/6,1/6,1/6)
  # min_val = L(min_par)
  # for (k in 1:100) {
  #   rand_vec <- runif(7)
  #   rand_vec <- rand_vec * (maximum - minimum) + minimum
  #
  #   curr_val <- L(rand_vec)
  #     if(curr_val < min_val) {
  #      min_val <- curr_val
  #       min_par <- rand_vec
  #     }
  # }

# print(min_par)
# print(L(min_par))
```

Competitive Test Set

```
#
# M <- nrow(testing_data)
# submission_df <- data.frame(matrix(ncol = 2, nrow = 0))
# colnames(submission_df) <-c("seq_id", "tm")
#
# problem_testing_samples = list()
#
# for (row in 1:M) {
#     tryCatch ( {
#       curr_ph <- testing_data[row, "pH"]
#       if(is.na(curr_ph)) {
#         curr_ph <- 7
#       }
#     }, error = function(e) {
#      #print(e)
#      #print(row)
#     problem_testing_samples = append(problem_testing_samples, row)
#     curr_ph <- 7
#    }
#    )
#     Y_pred = predictor(min_par,testing_data[row, "protein_sequence"],curr_ph)
#     submission_df[nrow(submission_df) + 1,] <- c(testing_data[row, "seq_id"],Y_pred)
# }
#
# write.csv(submission_df,"submission.csv", row.names = TRUE)
```

```
# print(unique(problem_training_samples))
# print(unique(problem_testing_samples))
```

Valuable Questions

- **How much of this data is explainable at this level?** Compared to other competitors using large scale language models like BERT, this model is incredible tiny. It's also makes a fundamental assumption that the sequence can be permutated without any effect on the output. This assumption significantly allows for complexity reduction. Does this model compete well with more advanced models?
- **Are some chain subsets more important than others?** Creating a model in this way, similar to those we have made all quarter long, allow us to analyze our coefficients and draw conclusions about the data our models describe. We might be able to make use of our coefficient analysis to be able to conclude either problematic unstable sequencing and/or good stable sequencing.

Datasource

Novozymes Enzyme Stability Prediction Competition Website:

`https://www.kaggle.com/competitions/novozymes-enzyme-stability-prediction/overview`