

Supplementary analysis description for manuscript: Unveiling the physical properties predictive of oil binding capacity in an interesterified palm-based fat

2023-11-09

Introduction

The goal of this work is to find parametric models that well approximate the oil binding capacity (i.e. obcc) of an interesterified palm-based fat given its other attributes. In this analysis, we consider 108 (3 dilutions x 4 crystallization conditions x 3 replicates x 3 time and storage points). The variables considered for inclusion in the models include: sfc, hard, g1, g2, delta, tpeak, enth, crysize, crynum, cryarea.

There are 32 of the 108 samples with missing values of hardness. In every case, the result of the missingness is that the fat's hardness fell below the minimal detection unit. This in mind, we impute all missing values of the fat's hardness with a nominal minimal value of 0.015 units, which is slightly smaller than the minimal measured value of hardness of 0.022.

```
# No missing values of hardness for the palm data so 108 remains the  
# number of observations.
```

```
p_fat_palm <- p_fat |>  
  tidyr::replace_na(list(hard = 0.015)) |>  
  filter(!is.na(obcc), !is.na(hard))
```

Many of the considered variables are highly skewed to the right, making it difficult to use standard regression techniques for modeling. We also observe that many of the variables share a highly-nonlinear relationship with the response variable (see Figure 1).

```
# Must have two line spacing for figure caps:
```

```
# - https://stackoverflow.com/questions/27444804/some-figure-captions-from-rmarkdown-n
```

```
p_fat_palm |>  
  select(-id, -fat) |>  
  pivot_longer(cols = !last_col()) |>  
  ggplot(aes(x = value, y = obcc)) +  
  geom_point() +  
  facet_wrap(~ name, scales = "free_x")
```

We observe that several variables share strong and non-linear relationships with obcc. By making the following transformations, we are able to make many of these relationships

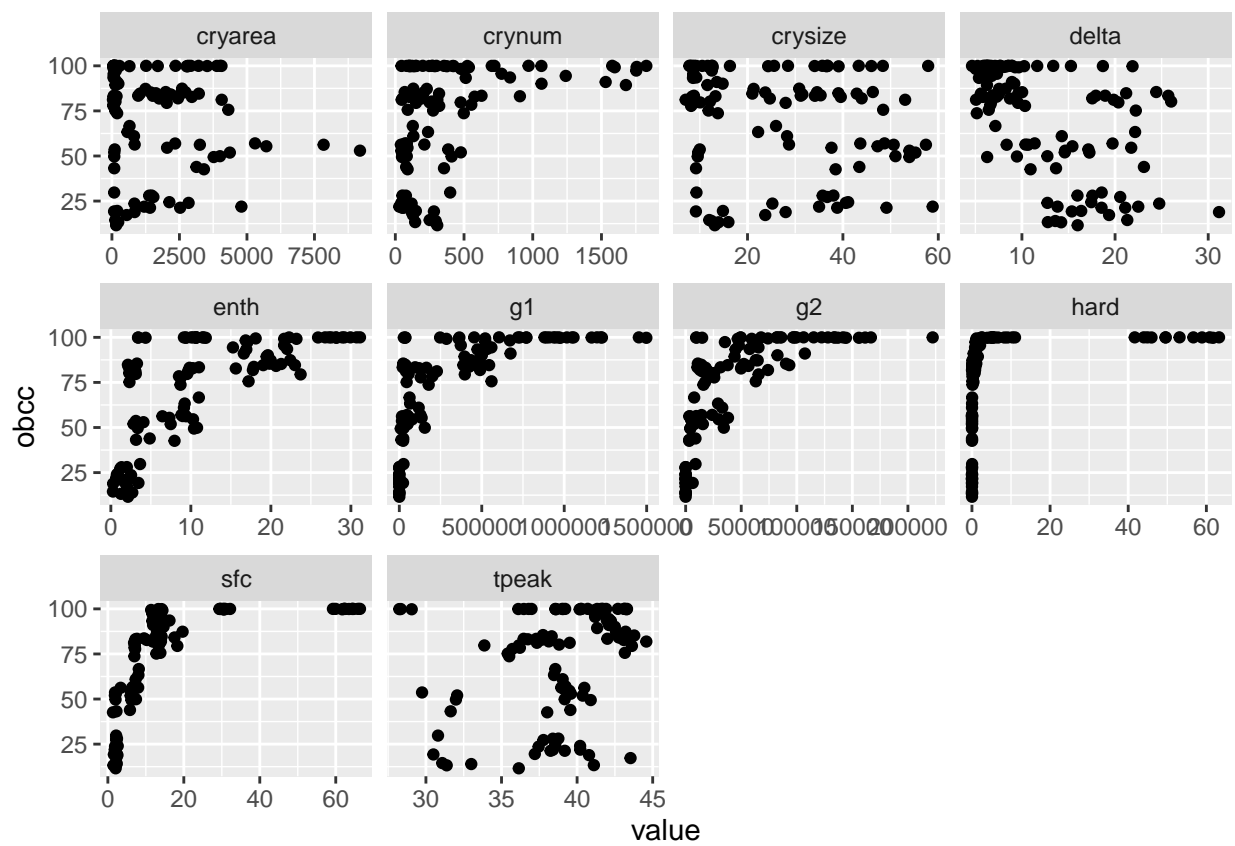


Figure 1: Scatterplots showing the relationship of each variable to obcc.

approximately linear. Note that we take log, double log, or square root, transformations of all variables except for `tpeak`. These transformations, as shown below, seem to make the pairwise relationships between each variable and `obcc` approximately linear (or non-existent), making this problem a candidate for multiple linear regression (see Figure 2).

```
# Note we transform all but one of the explanatory variables and only
# retain the transformed versions.
```

```
p_fat_palm_sub <- p_fat_palm |>
  mutate(lhard = log(log(hard + 1)), # Take a double log!
         lsfc = log(log(sfc + 1)),
         lg1 = sqrt(g1),
         lg2 = sqrt(g2),
         ldelta = log(delta),
         lenth = log(enth),
         lcrysize = log(crysize),
         lcryarea = log(cryarea),
         lcrynum = log(crynum)) |>
  select(lhard, lsfc, lg1, lg2, tpeak, lenth, ldelta,
         lcrynum, lcrysize, lcryarea, obcc)
```

```
p_fat_palm_sub |>
  pivot_longer(cols = !last_col()) |>
  ggplot(aes(x = value, y = obcc)) +
  geom_point() +
  facet_wrap(~ name, scales = "free_x")
```

However, one glaring issue with linear regression is the high degree of relatedness among the predictor variables. Figure 3 visualizes Spearman correlation coefficients among the candidate predictor variables. The Spearman correlation calculation uses a rank-based method of computing correlations, which makes it robust to non-linear relationships.

```
library(ggcorrplot)
tmat <- cor(p_fat_palm_sub[, -ncol(p_fat_palm_sub)], method = "spearman")
ggcorrplot::ggcorrplot(tmat,
                       method = "square", show.diag = FALSE, hc.order = TRUE,
                       tl.cex = 20, type = "lower", lab_size = 3,
                       lab = TRUE) +
  theme(text = element_text(size = 20),
        legend.key.height = unit(1, "cm"))
```

Many of these correlation coefficients have magnitudes (positive or negative) greater than 0.7, indicating high similarity in information. This property is known to obscure the interpretation of the coefficients in a linear regression model. For this reason, we elect to use LASSO regression.

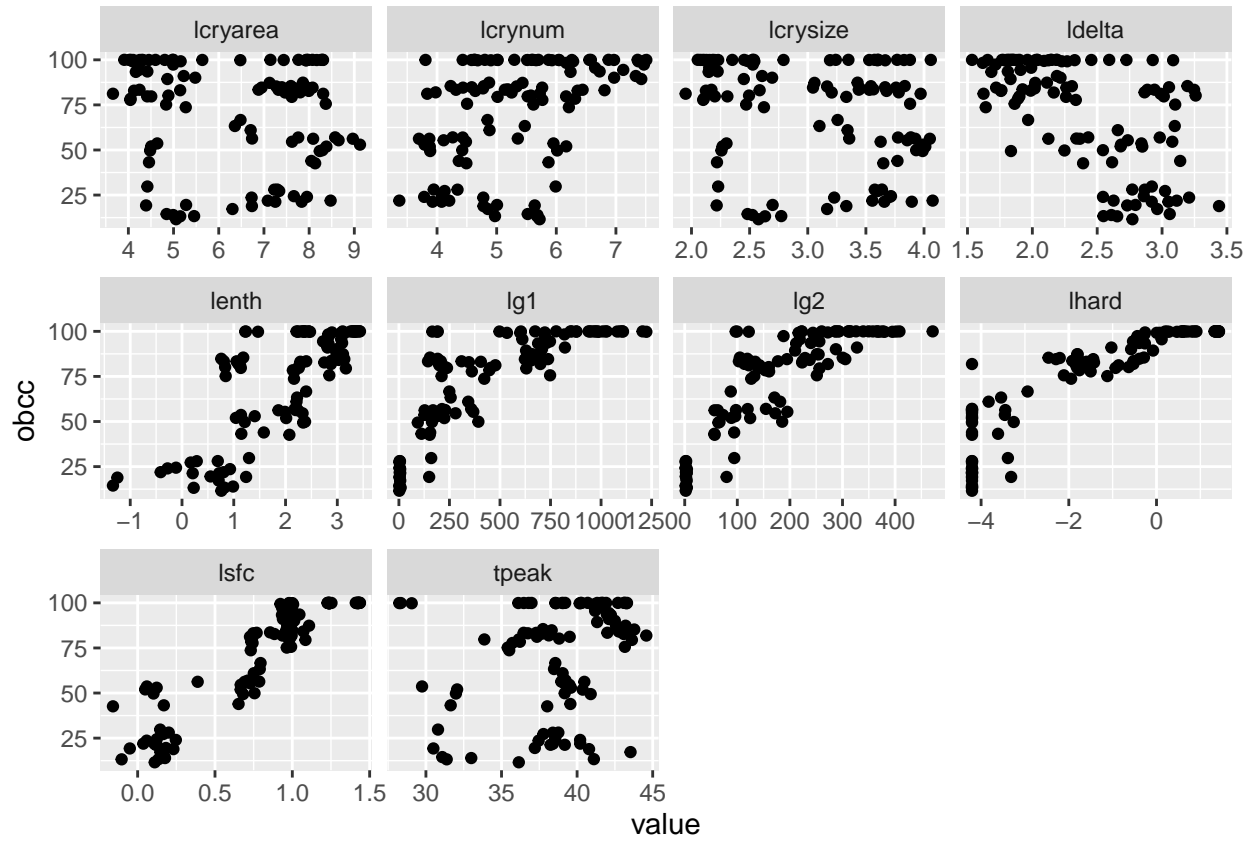


Figure 2: Scatterplots showing the relationship the transformed variables share with obcc.

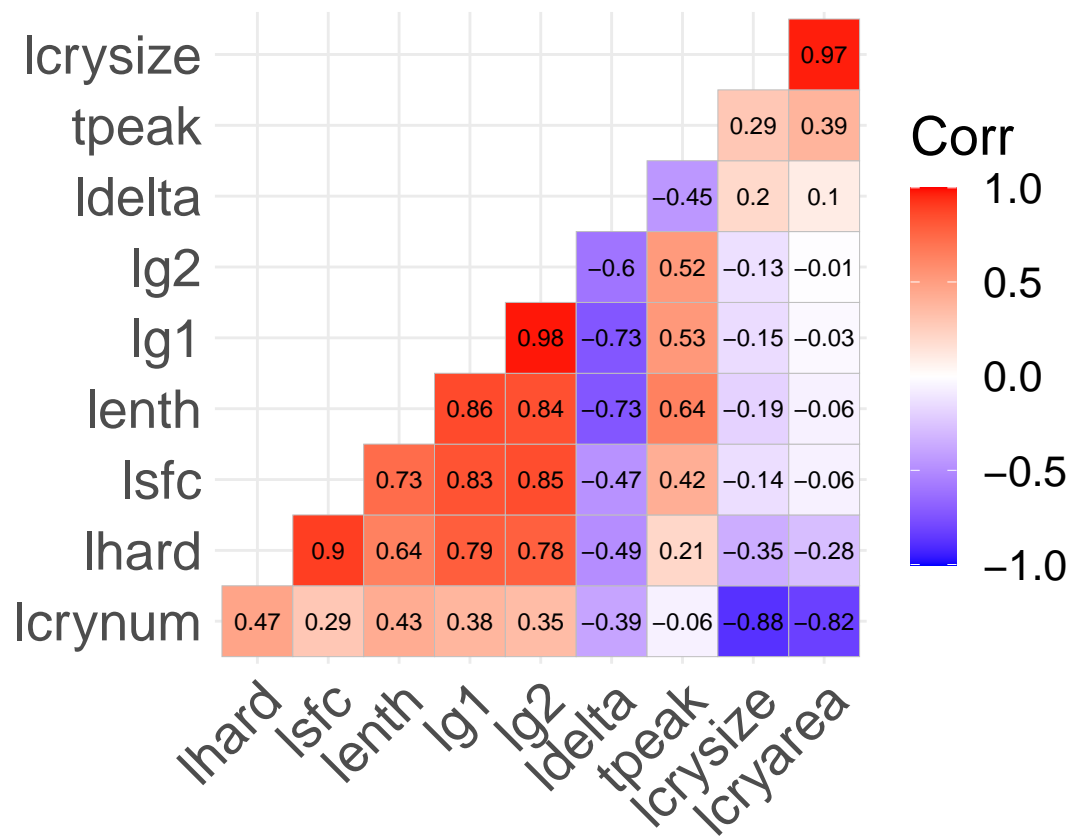


Figure 3: Plot of spearman correlation matrix.

LASSO Regression

LASSO regression uses a penalty term (λ) to discourage models with high magnitude coefficients. It tends to force some coefficient estimates to zero, making it a natural approach for variable selection.

In order to use LASSO, all variables need to have a similar magnitude, so that all regression coefficients are directly comparable to each other. If you do not scale, you overly penalize larger units of measurement (for example, without standardization, you would over-penalize measures in meters as opposed to centimeters). That in mind, the R software automatically standardizes variables during the calculation, then presents the coefficients on the original units.

Figure 4 shows the estimated coefficients for our model as the penalty decreases left to right. We notice that penalty quickly removes all but three variables from the model.

```
# Change to the last column.
x <- p_fat_palm_sub |>
  select(-obcc)

# glmnet does standardization automatically prior to fitting
# so we don't need to worry about doing it ourselves, but the coefficients
# are returned on the original units.
tmodel <- glmnet::glmnet(as.matrix(x), p_fat_palm_sub$obcc,
                        family = "gaussian",
                        alpha = 1,
                        standardize = TRUE)

plot(tmodel)
```

One way to determine the number of variables to retain is via cross-validation (CV). Here we use the 1-SE rule, which is to take the simplest model who has a CV result within one standard deviation of the absolute minimum for the model. This number changes slightly, so we re-run the cross validation result 50 times and retain the median lambda value satisfying the 1-se rule.

```
set.seed(702111)
tlam <- vector("numeric", 50)
for(i in seq_len(length(tlam))){
  tlam[i] <- glmnet::cv.glmnet(as.matrix(x), p_fat_palm_sub$obcc,
                             family = "gaussian",
                             alpha = 1)$lambda.1se
}

tlam_final <- median(tlam)
```

This results in a model with four coefficients and the model form:

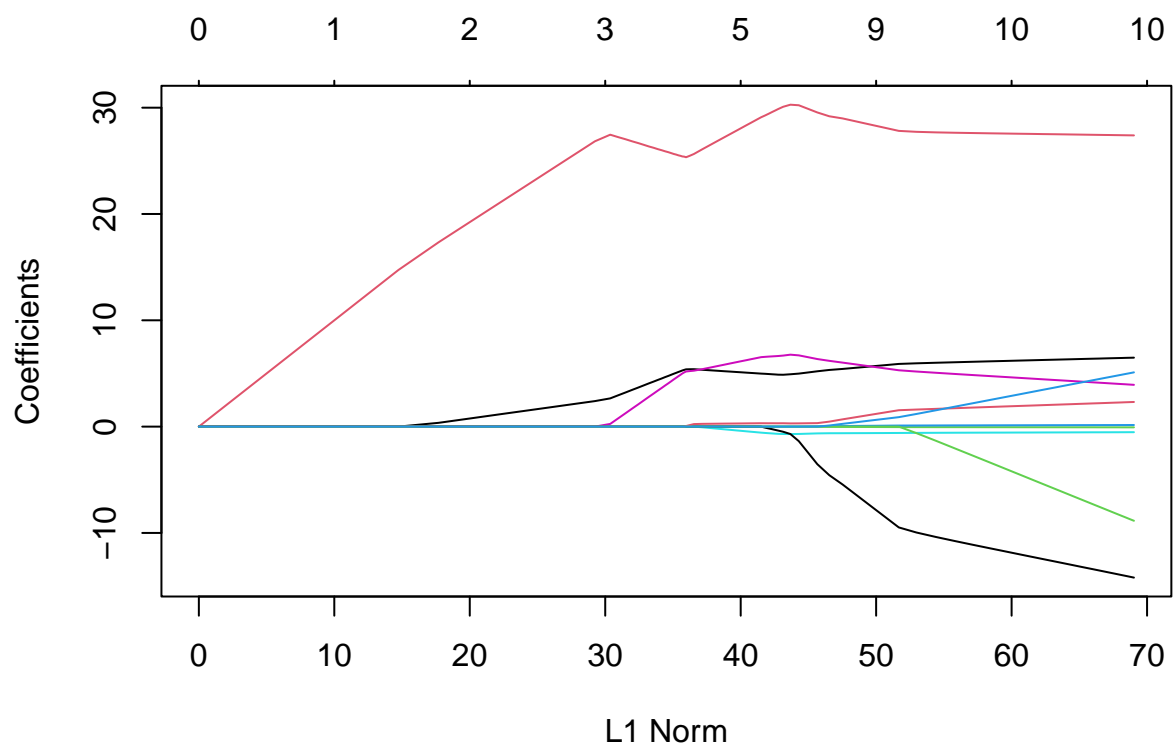


Figure 4: Trace plots of coefficient magnitudes as LASSO penalty increases.

```
tmodel <- glmnet::glmnet(x, p_fat_palm_sub$obcc,
                        family = "gaussian",
                        alpha = 1,
                        lambda = tlam_final)
```

```
tmodel$beta
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##              s0
## lhard      5.2019289
## lsfc      27.3395156
## lg1        .
## lg2        .
## tpeak     -0.2825988
## lenth      5.9042355
## ldelta     .
## lcrynum    0.2807553
## lcrysize   .
## lcryarea   .
```

$$\hat{y} = 5.2t_1(x_1) + 27.34t_1(x_2) - 0.28x_3 + 5.9t_2(x_4)^* + 0.28t_2(x_5)^* \quad (1)$$

where x_1 represents hardness, x_2 represents solid fat content, x_3 represents **tpeak**, x_4 represents **enth** and x_5 represents the crystal number. The functions $t_1(x) = \log(\log(x + 1))$ and $t_2(x) = \log(x)$ represent the two variable transformations used for the variables in question. This model, like all other models presented in this report, are necessarily bounded below by zero and above by 100.

Please note that the result is conditional upon the variables included in the model as well as the data provided for model training. As will be shown later, the model predicting **obcc** with hardness is essentially equivalent in accuracy to the same model using **sfc** to predict **obcc** as the two variables are so highly correlated.

Results

In addition to the penalized linear model using all coefficients, we also fit simple linear regression models for both hardness and sfc. These models take the form.

```
t1m1 <- lm(obcc ~ lhard, data = p_fat_palm_sub)
t1m2 <- lm(obcc ~ lsfc, data = p_fat_palm_sub)
```

$$\hat{y} = 83.59 + 11.83t(x_1) \quad (2)$$

$$\hat{y} = 16.66 + 60.85t(x_2) \quad (3)$$

We now compare the penalized regression model to these simplified models using CV. These models will be compared to along the null model, which simply predicts the average obcc of the training dataset. Note that the dataset includes three replicates for each sampling condition. Unlike the CV used for the lasso regression λ selection, this CV version separates the observations based on testing conditions. Thus, all three replicates of a specific experiment will always be contained within the same k-fold group of CV. The code for creating this version of CV, as well as the comparison on all of the models, is summarized below.

```
k <- 10
p_fat_palm_sub2 <- p_fat_palm_sub |>
  mutate(id = p_fat_palm$id,
         tid = gsub(id, pattern = "[[:space:]]+R[[:digit:]]+$",
                    replacement = ""))

# Create CV groups. Ended up just using group ID's (variant of LOOCV) instead
# of doing k-fold CV
tdf <- data.frame(tid = unique(p_fat_palm_sub2$tid))
tdf <- tdf |>
  dplyr::mutate(gid = seq_len(nrow(tdf)),
               cvid = rep(1:k, length.out = nrow(tdf)),
               cvid = sample(cvid)) |>
  dplyr::select(tid, cvid)

groups <- dplyr::left_join(p_fat_palm_sub2, tdf) |>
  pull(cvid)

## Joining with `by = join_by(tid)`

preds <- matrix(0, nrow = nrow(p_fat_palm_sub), ncol = 5)
for(i in seq_len(k)){
  train <- as.matrix(x[groups != i, ])
  resp <- p_fat_palm_sub$obcc[groups != i]
  test <- as.matrix(x[groups == i, ])

  tmodel1 <- glmnet::glmnet(train, resp,
                           family = "gaussian",
                           alpha = 1,
                           lambda = tlam_final)
  tmodel2 <- lm.fit(x = cbind(1, train[, "lhard"]), y = resp)
  tmodel3 <- lm.fit(x = cbind(1, train[, "lsfc"]), y = resp)
  tmodel4 <- lm.fit(x = cbind(1, train), y = resp)

  preds[groups == i, 1] <- predict(tmodel1, test)
  preds[groups == i, 2] <- cbind(1, test[, "lhard"]) %*% tmodel2$coefficients
  preds[groups == i, 3] <- cbind(1, test[, "lsfc"]) %*% tmodel3$coefficients
  preds[groups == i, 4] <- cbind(1, test) %*% tmodel4$coefficients
}
```

```

  preds[groups == i, 5] <- mean(resp)
}

preds <- pmin(pmax(preds, 0), 100)

rmse <- apply(preds, 2, function(x) sqrt(mean((x - p_fat_palm_sub$obcc)^2)))

rmse

## [1]  9.413797 13.453746 11.629677  9.137012 30.669662

```

The results show that the LASSO model has an RMSE of 9.41, the hardness-only model has an RMSE of 13.45, the sfc-only model has an RMSE of 11.63. All of these are compared to the null model which has an RMSE of 30.67. This implies that the multiple linear regression model reduces the error by about two-thirds.

Machine Learning Comparisons

We now compare all these results to some auto-tuned machine learning models using the **stressor** package in R. Note that this R package calls Python and does not use the same random number seeds as R does. This means that the results are slightly different each time this code is run. For this reason, we run this model once in ahead of time and save the results to call them later.

```

library(stressor)
# Code not run live since it takes a long time and a slightly complicated
# set up. But here is how I generate the table
tenv <- reticulate::virtualenv_list()
Sys.setenv(RETICULATE_PYTHON =
  paste0("C:/Users/brennan/Documents/.virtualenvs/",
    tenv[1],
    "/Scripts/python.exe"))
reticulate::use_virtualenv(tenv[1])
temp_models <- stressor::mlm_regressor(obcc ~ sfc + hard + g1 + g2 +
  delta + tpeak + enth + crysize + crynum +
  cryarea, data = p_fat_palm)

preds2 <- stressor::cv_core(temp_models, p_fat_palm, t_groups = groups)

final_ml <- rmse(preds2, p_fat_palm$obcc)

saveRDS(final_ml, file = "data-raw/stressor_table.RDS")

final_results <- readRDS(file = "data-raw/stressor_table.RDS")

```

final_results

##	models	rmse
## 1	et	6.618534
## 2	rf	6.546395
## 3	ada	6.782417
## 4	gbr	8.209345
## 5	lightgbm	8.695495
## 6	dt	11.023978
## 7	knn	12.949040
## 8	en	19.057144
## 9	llar	19.307748
## 10	lasso	19.307747
## 11	br	19.083983
## 12	lar	20.560845
## 13	ridge	19.623709
## 14	lr	19.633668
## 15	omp	22.312766
## 16	huber	23.245567
## 17	dummy	30.669662
## 18	par	137.910797

This shows that the the best machine learning models (extra trees regressor, random forest, and adaboost) using the untransformed variables do slightly better than the LASSO model, but not by much. Notice that LASSO and regression using the untransformed variables works very poorly for these data. The fact that the tree based methods do slightly better indicates the potential gain in accuracy that comes when modeling variable interactions. However, the LASSO model is quite attractive given that it has decent accuracy while also being much simpler than these complicated ML approaches.

Conclusions

LASSO regression provides a natural method for variable selection and controls the variance of the beta coefficients due to all the multi-collinearity in the data. Log and log-log transformations were critical to fashioning this problem into one suitable for regression. Our simple LASSO model (RMSE about 9.4) does better than the one-variable models using sfc (11.6) and hardness (13.5), but not as good as the best machine learning models (6.6). All methods do much better than the null model (30.7). Our recommendation would be the penalized linear regression model that makes use of four variables.

Appendix: Ideas that did not work

This section briefly explains approaches that did not work like we hoped so that we do not repeat them again.

Weibull model

Pairwise scatterplots (see below) revealed strong and non-linear relationships between many of the variables. The non-linear relationship shared among the explanatory variables make it difficult to isolate the effect of any single variable.

For this reason, we chose to proceed with a hierarchical modeling approach that focused on fitting a model with a subset of variables that had a (1) strong relationship with `obcc` and (2) were relatively easy to measure in a laboratory setting. For this reason, we chose to consider first a model using hardness (`hard`) to predict `obcc`.

```
ggplot(p_fat_palm) +  
  geom_point(aes(x = hard, y = obcc)) +  
  scale_x_log10(limits = c(0.01, 100)) +  
  theme_bw() +  
  theme(text = element_text(size = 16)) +  
  xlab("Hardness (log scale)") +  
  ylab("OBCC")
```

The pairwise scatterplot above shows that `hard` shares an asymptotic relationship with hardness. This motivates the use of asymptotic regression. We use the generalized version of Weibull regression to predict `obcc` (i.e. o_c) using `hard` (i.e. h_a):

$$o_{c_i} = \beta_0 + \beta_2 \exp \left(- \exp(\beta_3) h_{a_i}^{\beta_4} \right) + \epsilon_i \quad (4)$$

where $\beta_0 - \beta_4$ represent the coefficients that need to be trained with the data and ϵ represents the error term.

Note that this model form is not derived from a theoretical understanding of the relationship between o_c . Rather, this model was derived from visual explorations of the pairwise relationship between the two variables, as supplemented by one-dimensional smoothing splines. These splines, characterized as a one-dimensional generalized additive model (GAM), are provided by default as part of `ggplot2`'s `geom_smooth` function.

Note also that this model is not intended to diagnose the statistical significance of the relationship between hardness and `obcc`. It is obvious that the two variables are highly related to each other. The intent of the model is to derive a parametric approximation between the two variables and quantify the predictive accuracy.

```
tmodel <- nls(obcc ~ SSweibull(hard, b1, b2, b3, b4),  
             data = p_fat_palm)  
  
coef <- round(summary(tmodel)$coefficients[, 1], 3)
```

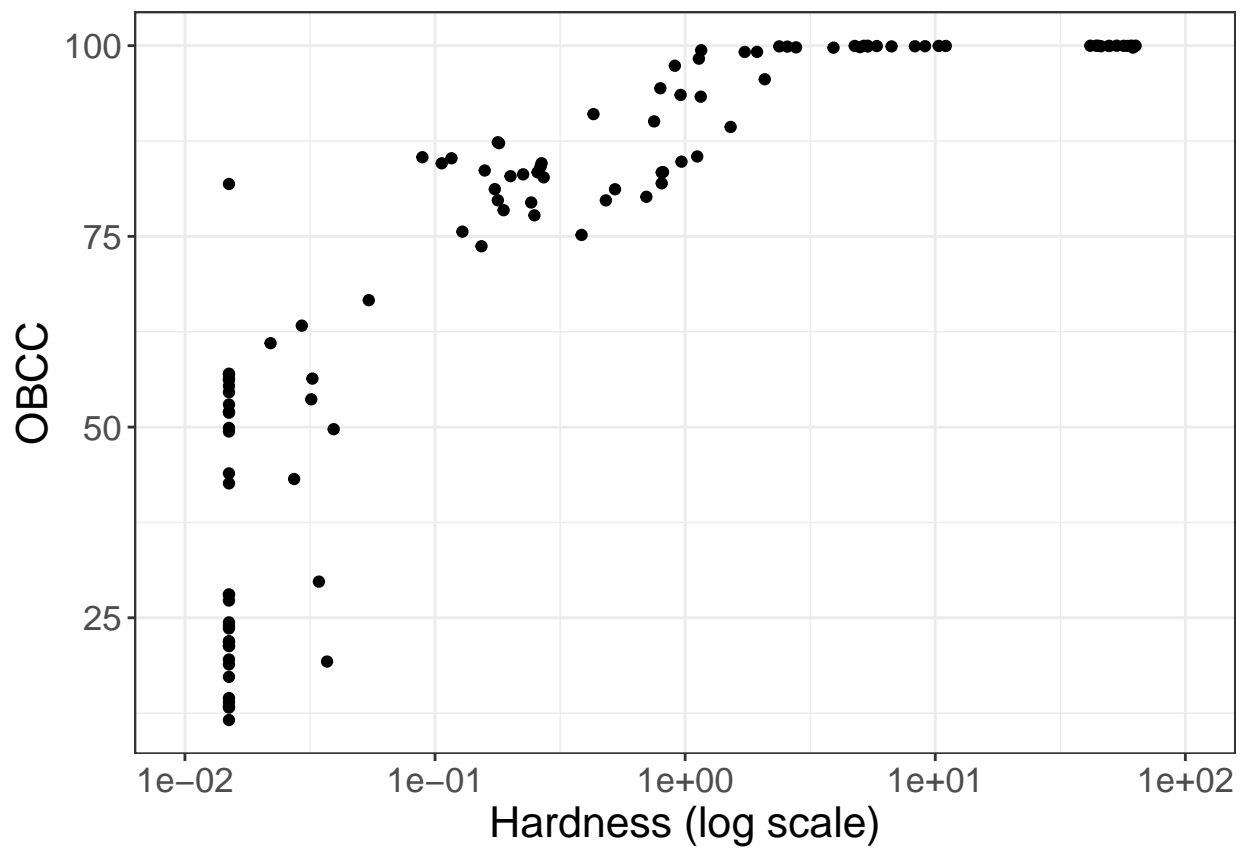


Figure 5: Scatterplot showing the relationship between hardness and obcc.

We use a self-starting Weibull model as part of a non-linear regression analysis in R to estimate the regression coefficients. The final model becomes.

$$\hat{o}_c = 101 + 2067.78 \exp \left(-\exp(1.795) h_a^{0.111} \right) \quad (5)$$

where \hat{o}_c represents the predicted value of obcc based on p_fat hardness. Note that predictions are capped above by 100 and below by zero due to practical constraints on the obcc metric.

The following figure shows the model fit, along with a plot of the residuals vs predicted values. This shows that the residuals are heteroskedastic. The residuals seem to be generally centered around zero, but there seems to be a potential pattern in the residuals (systematic over or under prediction for certain predicted values) that indicates the possibility of unexplained variation due to additional predictor variables.

```

preds <- predict(tmodel, data = p_fat_palm)
p_fat_palm <- p_fat_palm |>
  mutate(preds = preds,
         resid = obcc - preds)

null <- sum((p_fat_palm$obcc - mean(p_fat_palm$obcc))^2)
full <- sum(p_fat_palm$resid^2)

1 - (full / null)

## [1] 0.8378913

p1 <- ggplot(p_fat_palm) +
  stat_function(fun = function(x) coef[1] -
    coef[2] * exp(-exp(coef[3])*x^coef[4]),
    col = "grey", lwd = 2) +
  geom_point(aes(x = hard, y = obcc)) +
  scale_x_log10(limits = c(0.01, 100)) +
  theme_bw() +
  theme(text = element_text(size = 16)) +
  xlab("Hardness (log scale)") +
  ylab("OBCC")

p2 <- ggplot(p_fat_palm) +
  geom_point(aes(x = preds, y = resid)) +
  theme_bw() +
  theme(text = element_text(size = 16)) +
  xlab("Predicted OBCC") +
  ylab("Residuals") +
  geom_hline(yintercept = 0) +
  scale_y_continuous(limits = c(-50, 50))

```

```
gridExtra::grid.arrange(p1, p2, ncol = 2)
```

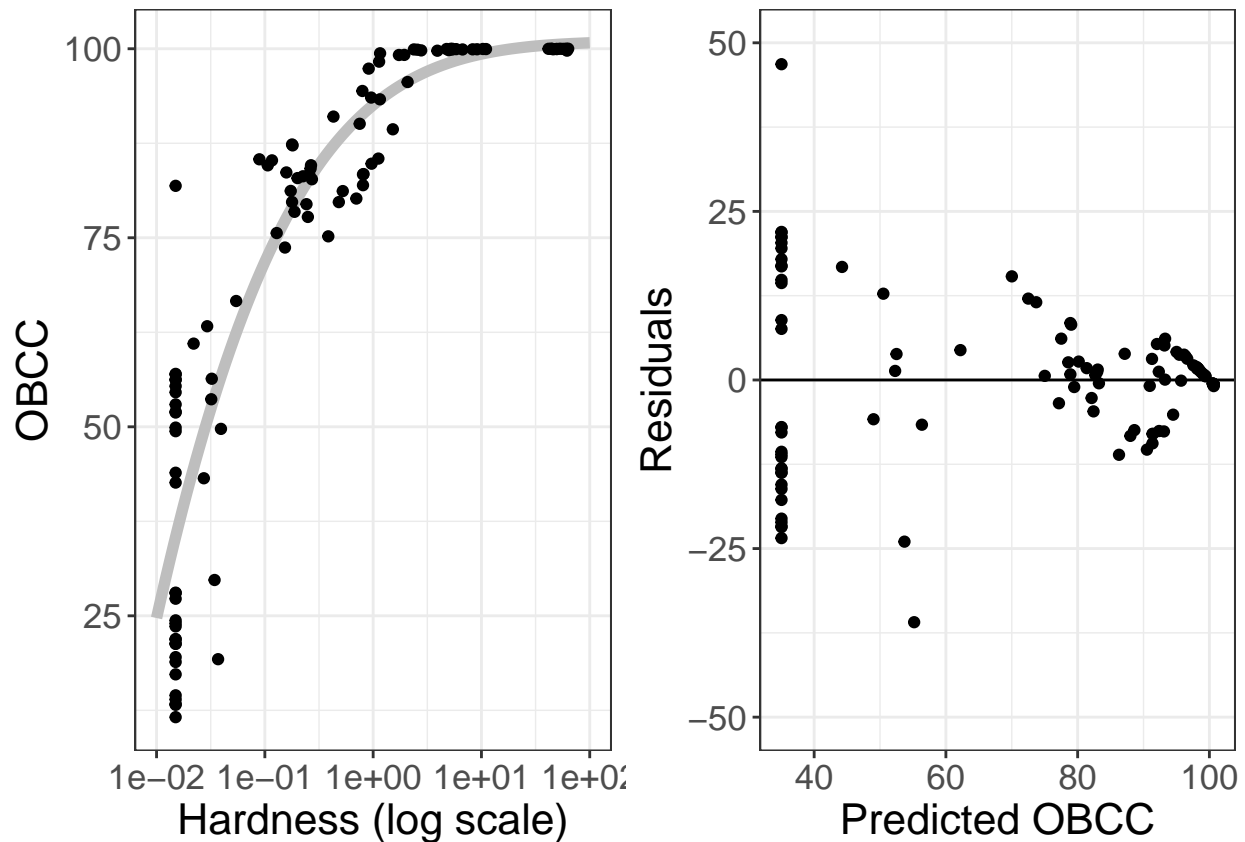


Figure 6: Visualization of asymptotic model results and its associated residuals.

We explore the potential for an improved model fit by considering various model fits to the residuals of the hardness model. We elect to do this because there are too few observations to simultaneously fit multiple Weibull models for the marginal effects.

While the Weibull model showed promise, attempts to perform CV failed miserably. The reason for the failure was that the numerical optimizer was not very robust to changes in the input data. Even leaving out a single group of the `p_fats` wreaked havoc on the ability to converge. This does not bode confidence in the proposed approach. Below is the code that was tested (along with many variations of the `nls` control parameters) that resulted in failure.

```
library(rlang) # For dynamic formula calls

# Separate p_fat by group.
p_fat_palm <- p_fat |>
  tidyr::replace_na(list(hard = 0.015)) |>
  dplyr::filter(!is.na(obcc), !is.na(hard)) |>
  dplyr::filter(fat == "palm") |>
  dplyr::mutate(tid = gsub(id, pattern = "[[:space:]]+R[[:digit:]]+$",
```

```

replacement = ""))

# Create CV groups. Ended up just using group ID's (variant of LOOCV) instead
# of doing k-fold CV
set.seed(902111)
tdf <- data.frame(tid = unique(p_fat_palm$tid)) |>
  dplyr::mutate(gid = seq_len(nrow(tdf)),
               cvid = rep(1:36, length.out = nrow(tdf)),
               cvid = sample(cvid)) |>
  dplyr::select(tid, gid)

p_fat_palm <- dplyr::left_join(p_fat_palm, tdf)

# Function to fit weibull model with any of the input variables/
asy_fun <- function(var, tdata, ...){
  # Rlang helps to inject the variable into the function.
  var = rlang::sym(var)
  nls(rlang::inject(obcc ~ SSweibull(!var, b1, b2, b3, b4)),
      data = tdata)
}

# Function only works with p_fat_palm dataframe as we have created it.
cv_asy <- function(var, tdata){
  k <- max(tdata$gid)

  preds <- vector("numeric", length = nrow(tdata))
  for(i in seq_len(k)){
    train <- tdata[tdata$gid != i, ]
    test <- tdata[tdata$gid == i, ]

    # Try function catches convergence error.
    tmodel <- tryCatch(asy_fun(var, train),
                      error = function(e) NULL)
    if(is.null(tmodel)){
      preds[tdata$gid == i] <- -1
    }else{
      preds[tdata$gid == i] <- as.numeric(predict(tmodel, test))
    }
  }

  preds
}

# Failed attempt to set the control parameters to fix the problem.

```



```
# tc <- nls.control(maxiter = 500, minFactor = 1e-4,
#                   scaleOffset = 1,
#                   nDcentral = TRUE)

pred1 <- cv_asy("hard", p_fat_palm)
```

GLM Standaridization

It turns out that glmnet does standardization under the hood, rendering this standardization table and the resulting transformation calculation useless.

Variable	$t_1(x_1)$	$t_1(x_2)$	$t_2(x_3)$	$t_2(x_4)$
$\mu_{t(x_i)}$	-1.14	0.88	2.13	5.64
$\sigma_{t(x_i)}$	2.07	0.42	1.24	1.25

Lastly, the ' indicates that the transformed variables have been standardized, meaning subtracting each variable by their respective mean $\mu_{t(x_i)}$ and dividing by the standard deviation $\sigma_{t(x_i)}$. The following table shows the mean and standard deviation for each of the considered variables.

```
tsum <- p_fat_palm_sub |>
  select(lhard, lsfc, lenth, lcrynum) |>
  pivot_longer(cols = everything()) |>
  group_by(name) |>
  summarize(mu = mean(value),
            sd = sd(value))
```

```
tsum
```

```
## # A tibble: 4 x 3
##   name      mu    sd
##   <chr>    <dbl> <dbl>
## 1 lcrynum  5.34  1.04
## 2 lenth   1.99  1.08
## 3 lhard   -1.70  2.07
## 4 lsfc    0.795 0.445
```