Predicting Chocolate Bar Ratings

Ryan Heslin

April 23, 2022

Introduction

This project's goal is to predict the ratings assigned by taste-testers to each of several thousand chocolate bars. Ratings are on the interval $[1, 4]$, in increments of 0.25, with higher ratings being better. Available features include bar ingredients, company and country of origin, cocoa percentage, and words used by testers to describe the bar.

The data come from a recent Tidy Tuesday and can be found here.

Available features include:

- `company_manufacturer`: Chocolate manufacturer
- `company_location`: Country of chocolate manufacturer.
- `country_of_bean_origin`: Country of cacao bean used in chocolate.
- `rating`: Taste-tester's rating of the chocolate bar, on a scale of 1 to 4 in increments of 0.25. The response.
- `ingredients`: Character vectors with abbreviations for the presence of six ingredients: cacao beans, cocoa butter, lecithin, sugar, salt, nonsugar sweetener, and vanilla. I transformed these into dummy columns.
- `characteristic`: Words and phrases used by taste-testers to describe the chocolate. I split this into five columns, one for each word.
- `review_date`: Year of review.
- `cocoa_percent`: Percentage of chocolate that is cocoa

Assumptions

From exploration, most of the characteristics seem to be descriptions of flavor (e.g., "sweet", "nutty"). If some of these flavors are more popular among taste-testers than others, they may be useful as a categorical variable. Company location and bean origin could prove useful by similar logic. If there was yearly turnover among the taste-testers, year might be worthwhile as well. There might also be a linear relationship between cocoa percentage and rating.

Preprocessing

As a first pass, I fit a simple linear regression predicting ratings. I don't expect it to do well, but it will provide a useful baseline. I create a "sentiment" feature by using `sentimentr` to compute sentiment for each phrase used to describe each chocolate, then summing the total for each chocolate bar. Positive values mean more positive sentiment. I don't simply compute sentiments for individual words because that would miscode negations (e.g., "not good").

It might have been better to extract this complex transformation to a function instead of jamming it into a recipe.

I also define my own recipe step, following the tutorial here. It takes a number of character vectors and returns dummy columns representing whether each of the n most common elements across all the columns is present in each row. This is a useful way of encoding the presence or absence of terms.

```r
suppressMessages(library(tidymodels))
library(tidyverse)
source(here::here("R", "utils.R"))
set.seed(12345)
theme_set(theme_minimal())
```

```r
chocolate_raw <- readRDS(here::here("data", "chocolate.Rds"))
chocolate_raw[["sentiment"]] <- NA
chocolate <- initial_split(chocolate_raw)
train <- training(chocolate)
test <- testing(chocolate)
```

These terms, as I noted earlier, are mostly flavor descriptors.

```r
select(train, starts_with("characteristic")) |>
  unlist(use.names = FALSE) |>
  table_head(.n = 6) |>
  vec2DF()
```

sweet	nutty	cocoa	roasty	earthy	creamy
218	190	185	163	141	128

```r
lin_spec <- linear_reg()
lin_rec <- recipe(train,
  formula = rating ~ .,
  vars = NULL
) |>
```

```
  step_mutate(
    sentiment = (across(starts_with("characteristic"), ~ str_remove_all(.x, "\\.") |>
      sentimentr::get_sentences() |>
      sentimentr::sentiment() |>
      pull("sentiment")) |>
      rowSums() |>
      unname()),
    across(where(is.logical), as.integer),
  ) |>
  step_term_dummy(starts_with("characteristic"), n = 10, match_terms = NULL, role = "predictor") |>
  step_other(company_location, country_of_bean_origin) |>
  step_dummy(c(all_nominal_predictors())) |>
  step_novel(all_nominal_predictors()) |>
  update_role(rating, new_role = "outcome") |>
  update_role(company_location, country_of_bean_origin, review_date,
    cocoa_percent, cocoa_butter, beans, lecithin, sugar, salt,
    nonsugar_sweetener, vanilla, sentiment,
    new_role = "predictor"
  ) |>
  update_role(ref, company_manufacturer, ingredients, specific_bean_origin_or_bar_name,
    starts_with("characteristic"),
    new_role = "id"
  )

lin_wflow <- workflow(lin_rec, lin_spec)
```

Customizing Prediction

I define a function (in `utils.R`) to round model predictions to the nearest increment on the scale. This violates the OLS assumptions, but this model is only intended as a reference. I use this function to develop a custom metric, `average_interval_error`, that measures each raw prediction's deviation from the true rating, rounded to the nearest increment, in units of the increment (0.25). For example, a prediction of 3.27 and a true rating of 2.5 calibrate to an interval error of 3 because $|2.5 - 3.27|$ rounded to the nearest fourth and multiplied by 4 (the inverse of the increment) is 3. Estimates below the scale's minimum are rounded up to it, and those above the maximum are rounded down to it. Stated mathematically, the average interval error is:

$$\text{AIE} = \frac{1}{n}\sum_{i=1}^{n} \min\left\{\max\left[\left|\frac{1}{\text{interval}}\left(y_i - \left\lfloor \hat{y}_i + \frac{1}{2}\right\rfloor\right)\right|, \min(y)\right], \max(y)\right\}$$

where $\lfloor x \rfloor = \text{floor}(x)$. (See here for an explanation of the formula).

Using this approach, mean absolute error is below one and a half increments - better than I expected. RMSE is somewhat higher, suggesting some errors are disproportionately large. The average absolute error is less than 2 intervals.

```
chocolate_metrics <- metric_set(mae, rmse, interval_error, rsq)
lin_model <- fit(lin_wflow, train)
tidy(lin_model)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-2.4644859	5.3464722	-0.4609555	0.6448842
review_date	0.0029458	0.0026590	1.1078616	0.2680644
cocoa_percent	-0.0087280	0.0017949	-4.8627961	0.0000013
beans	0.2861882	0.1284013	2.2288577	0.0259421
cocoa_butter	0.0458374	0.0217368	2.1087477	0.0350993
lecithin	-0.0542206	0.0267727	-2.0252194	0.0429870
sugar	0.0639667	0.1308955	0.4886851	0.6251221
salt	-0.0394498	0.0872382	-0.4522082	0.6511716
nonsugar_sweetener	-0.1332443	0.1510616	-0.8820529	0.3778618
vanilla	-0.1818966	0.0303367	-5.9959313	0.0000000
sentiment	0.2736841	0.0229305	11.9353571	0.0000000
sweet	-0.3937237	0.0336633	-11.6959176	0.0000000
nutty	0.0510471	0.0310573	1.6436420	0.1004186
cocoa	0.1764479	0.0305876	5.7686040	0.0000000
roasty	-0.0410449	0.0326045	-1.2588747	0.2082331
earthy	-0.1700608	0.0345277	-4.9253402	0.0000009
creamy	0.2285513	0.0365796	6.2480597	0.0000000
fatty	0.0047557	0.0387719	0.1226573	0.9023917
sandy	-0.0855217	0.0375928	-2.2749501	0.0230219
spicy	0.0963331	0.0381820	2.5229990	0.0117184
floral	-0.1611014	0.0433540	-3.7159491	0.0002084
company_location_France	0.0674320	0.0513326	1.3136308	0.1891320
company_location_U.K.	-0.1610706	0.0532746	-3.0234020	0.0025336
company_location_U.S.A.	-0.0685102	0.0372501	-1.8391950	0.0660453
company_location_other	-0.0615385	0.0382826	-1.6074810	0.1081181
country_of_bean_origin_Dominican.Republic	0.1470456	0.0492790	2.9839408	0.0028826
country_of_bean_origin_Ecuador	0.1143207	0.0503835	2.2690120	0.0233813
country_of_bean_origin_Madagascar	0.2072839	0.0504891	4.1055214	0.0000421
country_of_bean_origin_Peru	0.1181774	0.0491592	2.4039746	0.0163149
country_of_bean_origin_Venezuela	0.1464039	0.0480710	3.0455803	0.0023548
country_of_bean_origin_other	0.1083796	0.0411070	2.6365219	0.0084455

```
lin_preds <- augment(lin_model, new_data = train)
lin_preds[[".pred"]] <- fit_to_scale(lin_preds[[".pred"]], min = 1, max = 4, increment = .25)
chocolate_metrics(lin_preds, truth = rating, estimate = .pred)
```

.metric	.estimator	.estimate
mae	standard	0.2992884
rmse	standard	0.3953889
interval_error	standard	1.1971534
rsq	standard	0.2131016

Some of the flavor dummies appear to be important. According to the model, bars described as “sweet” are rated about 1.5 increments less than otherwise identical bars that are not “sweet”, all else held constant.

```
tidy(lin_model) |>
  arrange(-abs(estimate))
```

term	estimate	std.error	statistic	p.value
(Intercept)	-2.4644859	5.3464722	-0.4609555	0.6448842
sweet	-0.3937237	0.0336633	-11.6959176	0.0000000
beans	0.2861882	0.1284013	2.2288577	0.0259421
sentiment	0.2736841	0.0229305	11.9353571	0.0000000
creamy	0.2285513	0.0365796	6.2480597	0.0000000
country_of_bean_origin_Madagascar	0.2072839	0.0504891	4.1055214	0.0000421
vanilla	-0.1818966	0.0303367	-5.9959313	0.0000000
cocoa	0.1764479	0.0305876	5.7686040	0.0000000
earthy	-0.1700608	0.0345277	-4.9253402	0.0000009
floral	-0.1611014	0.0433540	-3.7159491	0.0002084
company_location_U.K.	-0.1610706	0.0532746	-3.0234020	0.0025336
country_of_bean_origin_Dominican.Republic	0.1470456	0.0492790	2.9839408	0.0028826
country_of_bean_origin_Venezuela	0.1464039	0.0480710	3.0455803	0.0023548
nonsugar_sweetener	-0.1332443	0.1510616	-0.8820529	0.3778618
country_of_bean_origin_Peru	0.1181774	0.0491592	2.4039746	0.0163149
country_of_bean_origin_Ecuador	0.1143207	0.0503835	2.2690120	0.0233813
country_of_bean_origin_other	0.1083796	0.0411070	2.6365219	0.0084455
spicy	0.0963331	0.0381820	2.5229990	0.0117184
sandy	-0.0855217	0.0375928	-2.2749501	0.0230219
company_location_U.S.A.	-0.0685102	0.0372501	-1.8391950	0.0660453
company_location_France	0.0674320	0.0513326	1.3136308	0.1891320
sugar	0.0639667	0.1308955	0.4886851	0.6251221
company_location_other	-0.0615385	0.0382826	-1.6074810	0.1081181
lecithin	-0.0542206	0.0267727	-2.0252194	0.0429870
nutty	0.0510471	0.0310573	1.6436420	0.1004186
cocoa_butter	0.0458374	0.0217368	2.1087477	0.0350993
roasty	-0.0410449	0.0326045	-1.2588747	0.2082331
salt	-0.0394498	0.0872382	-0.4522082	0.6511716
cocoa_percent	-0.0087280	0.0017949	-4.8627961	0.0000013
fatty	0.0047557	0.0387719	0.1226573	0.9023917
review_date	0.0029458	0.0026590	1.1078616	0.2680644

It seems the biggest errors are for the very worst chocolate.

```
range(lin_preds[[".pred"]])
```

```
[1] 2.25 4.00
```

```
arrange(lin_preds, -abs(.pred - rating)) |>
  head() |>
  select(company_manufacturer, specific_bean_origin_or_bar_name, .pred, rating)
```

company_manufacturer	specific_bean_origin_or_bar_name	.pred	rating
Callebaut	Baking	3.00	1.0
Neuhaus (Callebaut)	Dark	3.00	1.0
Cote d' Or (Kraft)	Sensations Intense	2.75	1.0
Machu Picchu Trading Co.	Peru	3.00	1.5
Cacaoyere (Ecuatoriana)	Pichincha	3.00	1.5
Escazu	Guapiles	3.25	2.0

Next, I try a random forest, reasoning that the relatively large number of predictors and likelihood of nonlinear associations suit it well for these data.

```r
baked <- prep(lin_rec) |> bake(train)
combined <-
  left_join(baked, select(
    train,
    all_of(setdiff(union(
      colnames(train),
      lin_rec[[c("var_info", "variable")]]
    ), colnames(baked))), ref
  ),
  by = "ref"
  )
folds <- vfold_cv(combined, v = 20)

rf_spec <- rand_forest(mtry = tune(), trees = 400, min_n = tune()) |>
  set_engine(engine = "ranger", importance = "permutation") |>
  set_mode("regression")
rf_wf <- workflow(lin_rec, rf_spec)
```

I use 20-fold cross validation to select the `mtry` and `min_n` parameters, using `average_interval_error` to select tuned values.

```r
suppressMessages(library(tidymodels))
mtry_start <- floor(nrow(tidy(lin_model)) / 3)
tuning_grid <- grid_regular(
  mtry(range = c(mtry_start - 2, mtry_start + 4)),
  min_n(range = c(5, 40))
)
tune_res <- tune_grid(rf_wf,
  grid = tuning_grid, resamples = folds,
  metrics = metric_set(interval_error)
)

rf_wf <- finalize_workflow(rf_wf, select_best(tune_res, "interval_error"))
rf_model <- fit(rf_wf, data = train)
rf_model
```

```
== Workflow [trained] ==========================================================
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor ----------------------------------------------------------------------------------
5 Recipe Steps

* step_mutate()
* step_term_dummy()
* step_other()
* step_dummy()
* step_novel()

-- Model -----------------------------------------------------------------------------------------
Ranger result

Call:
 ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~14L,      x), num.trees = ~400, min.no

Type:                             Regression
Number of trees:                  400
Sample size:                      1897
Number of independent variables:  30
Mtry:                             14
Target node size:                 5
Variable importance mode:         permutation
Splitrule:                        variance
OOB prediction error (MSE):       0.1512018
R squared (OOB):                  0.2349718
```

```
rf_preds <- predict(rf_model, new_data = train)
```

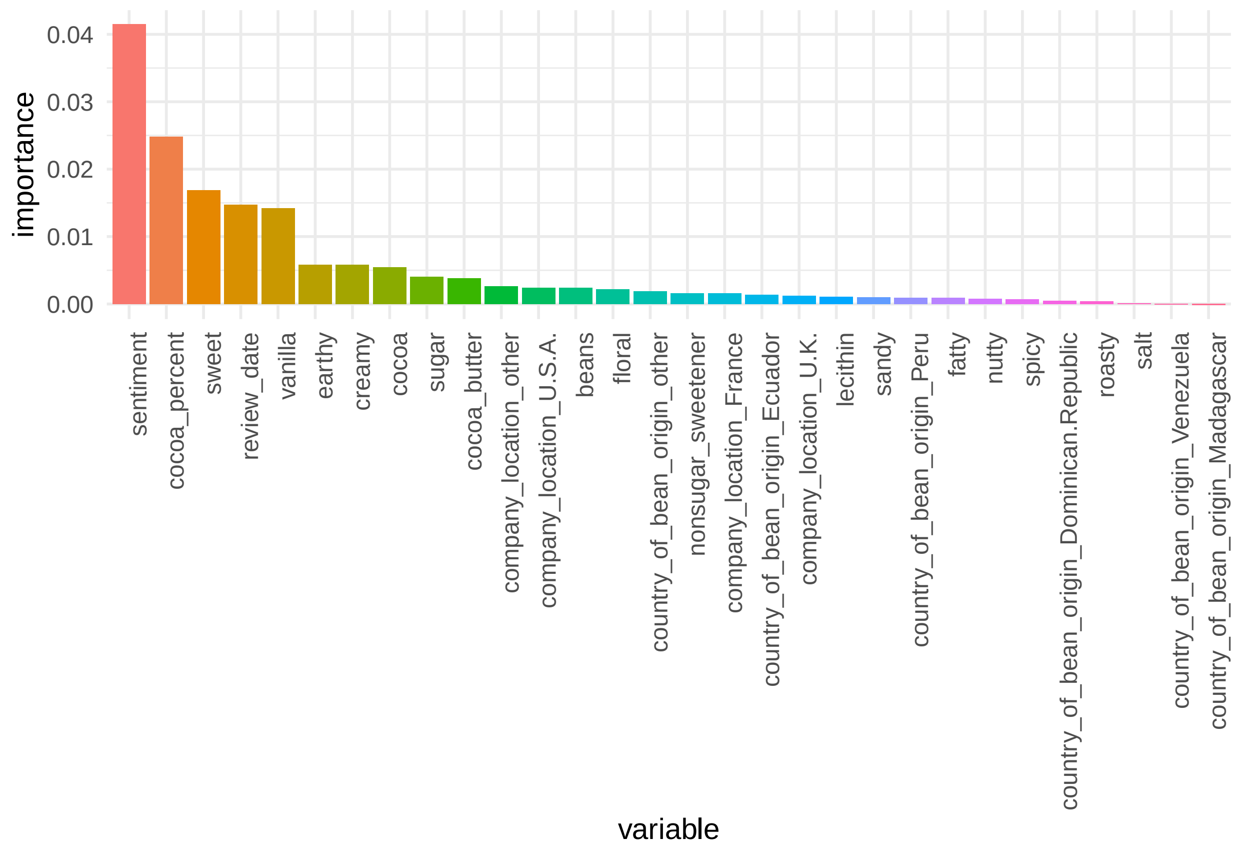
These predictions have a much higher range. MAE and RMSE are less than half what they were in regression

```
rf_preds <- augment(rf_model, new_data = train)
rf_preds[[".pred"]] <- fit_to_scale(rf_preds[[".pred"]], min = 1, max = 4, increment = .25)
chocolate_metrics(rf_preds, truth = rating, estimate = .pred)
```

.metric	.estimator	.estimate
mae	standard	0.1395625
rmse	standard	0.2073536
interval_error	standard	0.5582499
rsq	standard	0.8193373

Sentiment is by far the most important variable, followed by some of the ingredient dummies. The country of origin dummies don't seem that important.

```r
importance_bar <- rf_model |>
  extract_fit_engine() |>
  ranger::importance() |>
  enframe(name = "variable", value = "importance") |>
  mutate(variable = factor(variable,
    levels = variable[order(importance, decreasing = TRUE)]
  )) |>
  ggplot(aes(x = variable, y = importance)) +
  geom_col(aes(fill = variable), show.legend = FALSE) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
importance_bar
```



```r
ggsave(here::here("outputs", "importance_bar.png"), importance_bar)
```

I'm concerned about overfitting, so I also try boosting, which is less prone to overfitting.

```r
boost_spec <- boost_tree(
  mode = "regression", learn_rate = tune(),
  mtry = tune(),
  min_n = tune(), tree_depth = 15, sample_size = 1
)
```

```r
suppressMessages(library(tidymodels))
boost_wf <- workflow(lin_rec, boost_spec)
tuning_grid <- grid_regular(
  mtry(range = c(mtry_start - 2, mtry_start + 4)),
  learn_rate(),
  min_n()
)
tune_res <- tune_grid(boost_wf,
  grid = tuning_grid,
  resamples = folds, metrics = metric_set(interval_error)
)
boost_wf <- finalize_workflow(boost_wf, select_best(tune_res, "interval_error"))
boost_model <- fit(boost_wf, data = train)
boost_model
```

```
== Workflow [trained] =================================================================
Preprocessor: Recipe
Model: boost_tree()

-- Preprocessor -----------------------------------------------------------------------
5 Recipe Steps

* step_mutate()
* step_term_dummy()
* step_other()
* step_dummy()
* step_novel()

-- Model ------------------------------------------------------------------------------
##### xgb.Booster
raw: 55.9 Kb
call:
  xgboost::xgb.train(params = list(eta = 0.1, max_depth = 15, gamma = 0,
    colsample_bytree = 1, colsample_bynode = 0.466666666666667,
    min_child_weight = 2L, subsample = 1, objective = "reg:squarederror"),
    data = x$data, nrounds = 15, watchlist = x$watchlist, verbose = 0,
    nthread = 1)
params (as set within xgb.train):
  eta = "0.1", max_depth = "15", gamma = "0", colsample_bytree = "1", colsample_bynode = "0.46666666666
xgb.attributes:
  niter
callbacks:
  cb.evaluation.log()
# of features: 30
niter: 15
nfeatures : 30
evaluation_log:
    iter training_rmse
       1      2.469320
       2      2.230198
---
      14      0.724771
      15      0.669029
```

```
boost_preds <- predict(boost_model, new_data = train)
```

The boosted model does terribly compared to the others, with far greater errors.

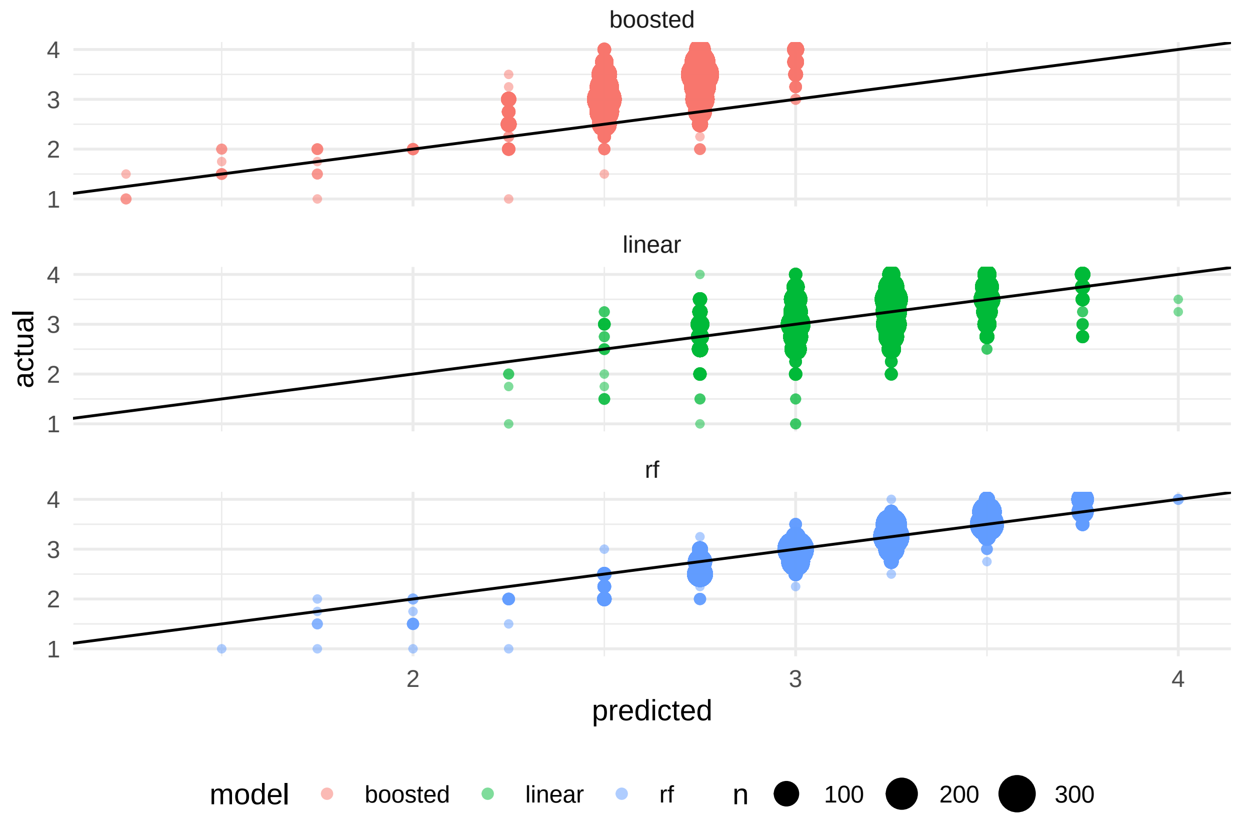
```
boost_preds <- augment(boost_model, new_data = train)
boost_preds[[".pred"]] <- fit_to_scale(boost_preds[[".pred"]], min = 1, max = 4, increment = .25)
chocolate_metrics(boost_preds, truth = rating, estimate = .pred)
```

.metric	.estimator	.estimate
mae	standard	0.5869794
rmse	standard	0.6773600
interval_error	standard	2.3479178
rsq	standard	0.3504098

Comparison

This plot clarifies the pattern

```
overall <- tibble(
  actual = train[["rating"]], linear = lin_preds[[".pred"]],
  rf = rf_preds[[".pred"]], boosted = boost_preds[[".pred"]]
)
fitted_actual_plot(overall, -actual)
```



Evaluation and Results

Now I predict on the test set. The boosted model does as terribly as before. Linear regression performs about the same as the random forest. The random forest's better performance on the training set was probably a result of overfitting.

```
lin_test <- test_preds(lin_model, test)
rf_test <- test_preds(rf_model, test)
boost_test <- test_preds(boost_model, test)

results <- bind_rows(
  linear = chocolate_metrics(lin_test, estimate = .pred, truth = rating),
  rf = chocolate_metrics(rf_test, estimate = .pred, truth = rating),
  boost = chocolate_metrics(boost_test, estimate = .pred, truth = rating),
  .id = "model"
)
results
```

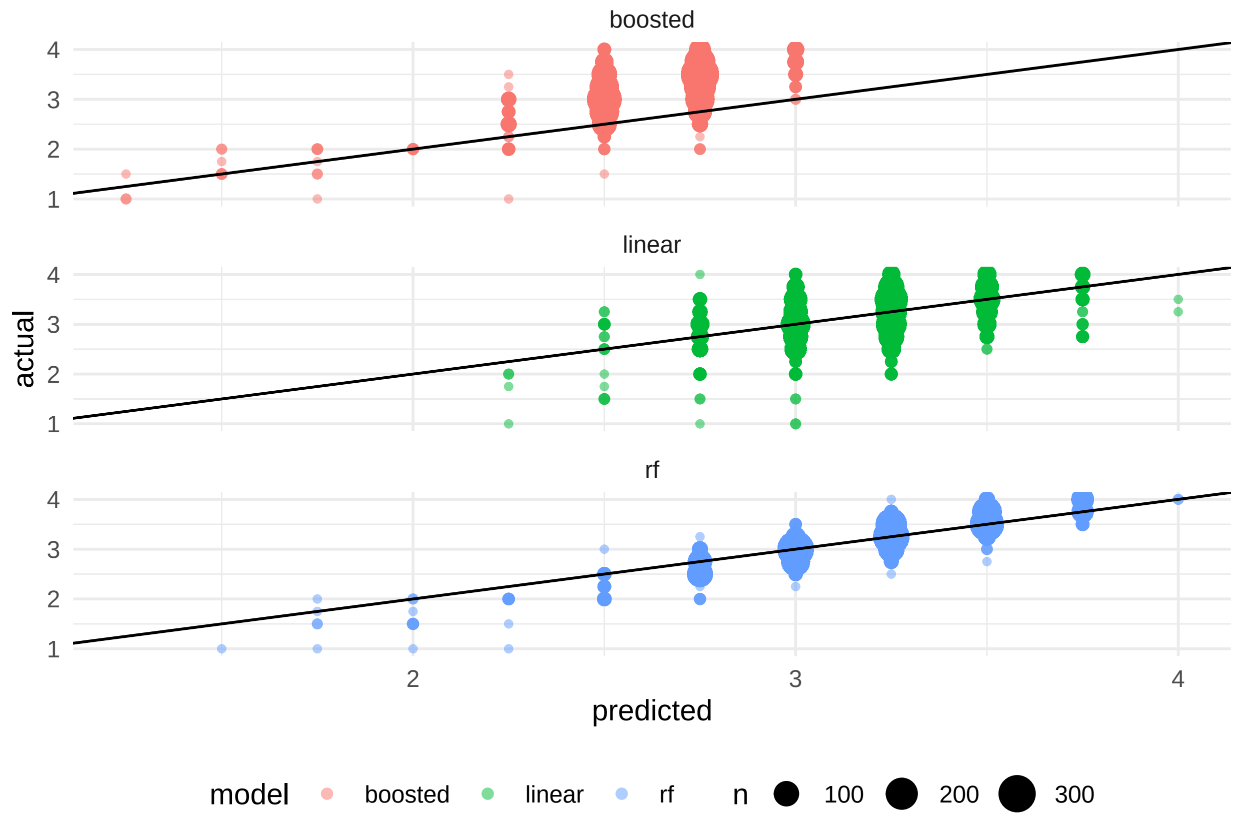
model	.metric	.estimator	.estimate
linear	mae	standard	0.3079662

model	.metric	.estimator	.estimate
linear	rmse	standard	0.3970624
linear	interval_error	standard	1.2211690
linear	rsq	standard	0.2176920
rf	mae	standard	0.3134230
rf	rmse	standard	0.4021690
rf	interval_error	standard	1.2527646
rf	rsq	standard	0.2092209
boost	mae	standard	0.5831447
boost	rmse	standard	0.6752469
boost	interval_error	standard	2.3270142
boost	rsq	standard	0.2039640

```r
write_csv(results, here::here("outputs", "results.csv"))
```

The same patterns appear when plotting the test predictions. The boosted model seems to be biased low. The random forest predicts in a wider range than the linear model, but has comparable performance by overall error metrics. I would prefer the linear model here because it is much simpler and easier to interpret.

```r
test_overall <- tibble(
  actual = test[["rating"]], linear = lin_test[[".pred"]],
  rf = rf_test[[".pred"]], boosted = boost_test[[".pred"]],
  ref = test[["ref"]]
)
fitted_actual <- fitted_actual_plot(overall, -actual)
fitted_actual
```



```
ggsave(here::here("outputs", "fitted-actual.png"), fitted_actual)
```

Again, the low-rated bars confounded the models the most.

```
left_join(test_overall, test, by = "ref") |>
  select(specific_bean_origin_or_bar_name, actual, linear, rf, boosted) |>
  arrange(-abs(linear - actual)) |>
  head()
```

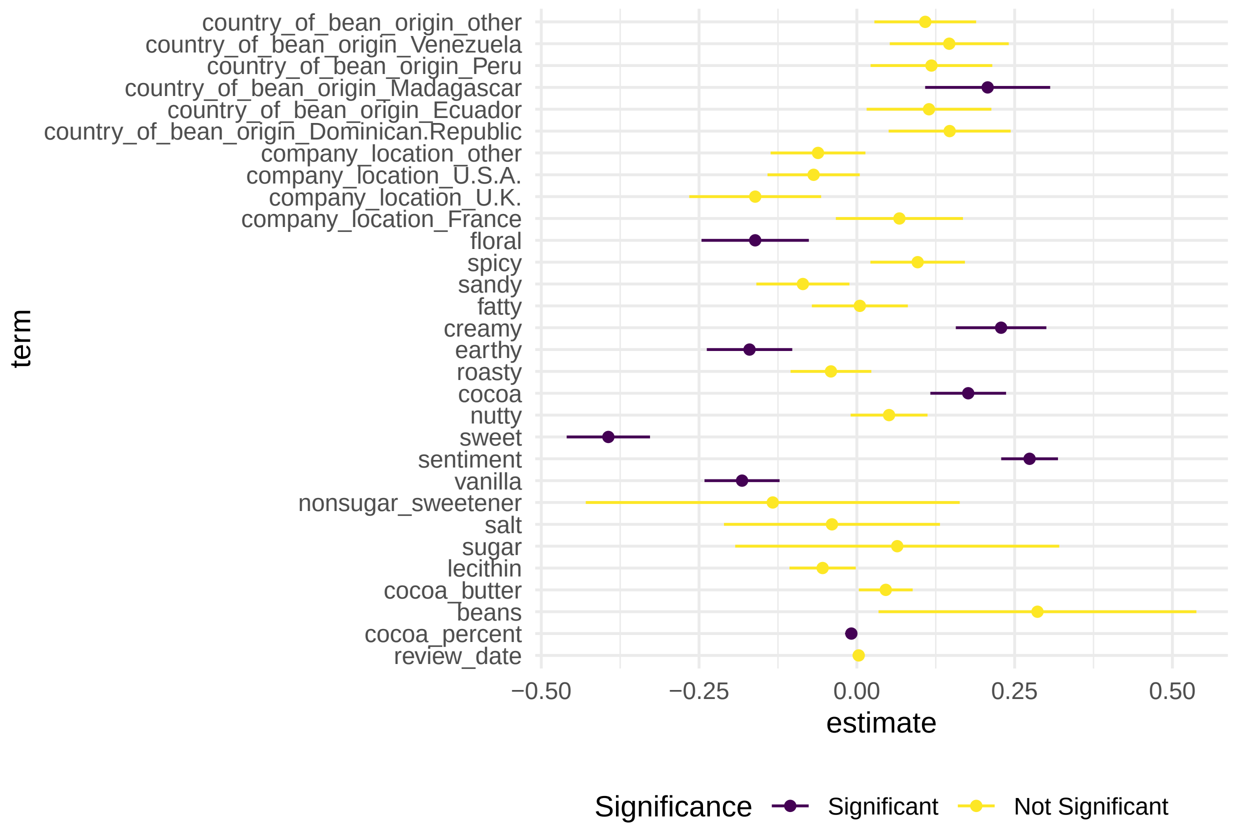
specific_bean_origin_or_bar_name	actual	linear	rf	boosted
Ghana	1.50	3.407396	3.493872	2.778029
Ghana	1.50	3.407396	3.493872	2.778029
Brazil Rio Doce	1.75	3.408195	3.376808	2.764607
Carenero Superior, Gran Saman	2.00	3.430857	2.961837	2.433774
Jamaique	2.00	3.430857	2.961837	2.433774
Colombian Semi Dark	2.00	3.430857	2.961837	2.433774

Most of the estimated coefficients from the linear model are not significant at $\alpha = .05$ after applying the Bonferroni correction. Some standard errors are large, likely due to there being few observations with that factor level.

```r
tidied <- tidy(lin_model) |>
  filter(term != "(Intercept)") |>
  mutate(
    term = factor(term, levels = term),
    Significance = factor(c(
      "Significant",
      "Not Significant"
    )[as.integer(p.adjust(p.value,
      method = "bonferroni"
    ) > .05) + 1], levels = c("Significant", "Not Significant"))
  )

width <- qt(1 - .025, df = nrow(train) - nrow(tidied))

coef_point <- ggplot(tidied, aes(x = estimate, color = Significance)) +
  geom_point(aes(y = term)) +
  geom_segment(aes(
    yend = seq_len(nrow(tidied)),
    y = seq_len(nrow(tidied)), xend = estimate - width * std.error, x = estimate + width * std.error
  )) +
  scale_color_viridis_d() +
  theme(legend.position = "bottom")
coef_point
```



```r
ggsave(here::here("outputs", "coef_point.png"), coef_point)
```

Conclusion

The presence or absence of flavors identified by taste-testers, as well as the overall text sentiment of their descriptions, had a substantial relationship with chocolate bar ratings. As expected, positive sentiment was associated with higher ratings. Some other features, such as country of bean origin, were marginally useful, though not all categories proved important in either linear regression or the random forest model. It is hard to tell from these observations exactly how these factors influence chocolate ratings, but ingredient and characteristic variables could be used. Constructing features based on sentiment took some work, but was well worth doing. I also took the opportunity to delve into the `tidymodels` interface by defining a custom metric and recipe step. I probably could have gotten by using the built-in tools, but I enjoyed customizing them.

I think I caused myself problems by storing the words used to describe each chocolate bar in five atomic vector columns. A single list column would have made more conceptual sense and required less work to transform for analysis. I also could have done more to fine-tune the random forest model, or perhaps experimented with KNN after using the lasso, or PCA to reduce the number of variables. Another unexplored option would have been further condensing the levels of categorical variables, since many of these features did not prove very useful. Still, I think my final models were reasonably successful.

After completing modeling, I read Julia Silge's report on predicting with these data. She tokenized words from the text descriptions, computed tf-idfs, and used them as features in a support vector machine model, which I did not think of trying, and obtained good results, beating my test RMSE.

References

Create your own recipe step function. (n.d.). Retrieved April 23, 2022, from https://www.tidymodels.org/learn/develop/recipes/

Custom performance metrics. (n.d.). Retrieved April 23, 2022, from https://www.tidymodels.org/learn/develop/metrics/

Text predictors for #TidyTuesday chocolate ratings. (n.d.). Julia Silge. Retrieved April 23, 2022, from https://juliasilge.com/blog/chocolate-ratings/