Chocolate Modeling

Ryan Heslin

April 13, 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.

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 higher 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 the each of the n most common elements across all the columns is present in each row.

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

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

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, `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 $\lfloor\ \rfloor$ designates the floor function. (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 equal to about 1.3 intervals.

```
chocolate_metrics <- metric_set(mae, rmse, interval_error)
baked <- prep(lin_rec) |>
  bake(train)
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

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

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

```
[1] 2.25 4.00
```

```r
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

The most common words pertain to flavors, not sentiments. These might be useful as categorical features

```r
select(train, starts_with("characteristic")) |>
  unlist(use.names = FALSE) |>
  table_head(.n = 200) |>
  vec2DF()
```

sweet	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
218	190	185	163	141	128	125	120	114	110	108	106	68	66	65	64	61	60	57	55	51	49	48	48	46	45	45	42	40	40	40	38	38	32	32	31	30	30	29	27	27	25	25	24	24	2

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.

```
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)
```

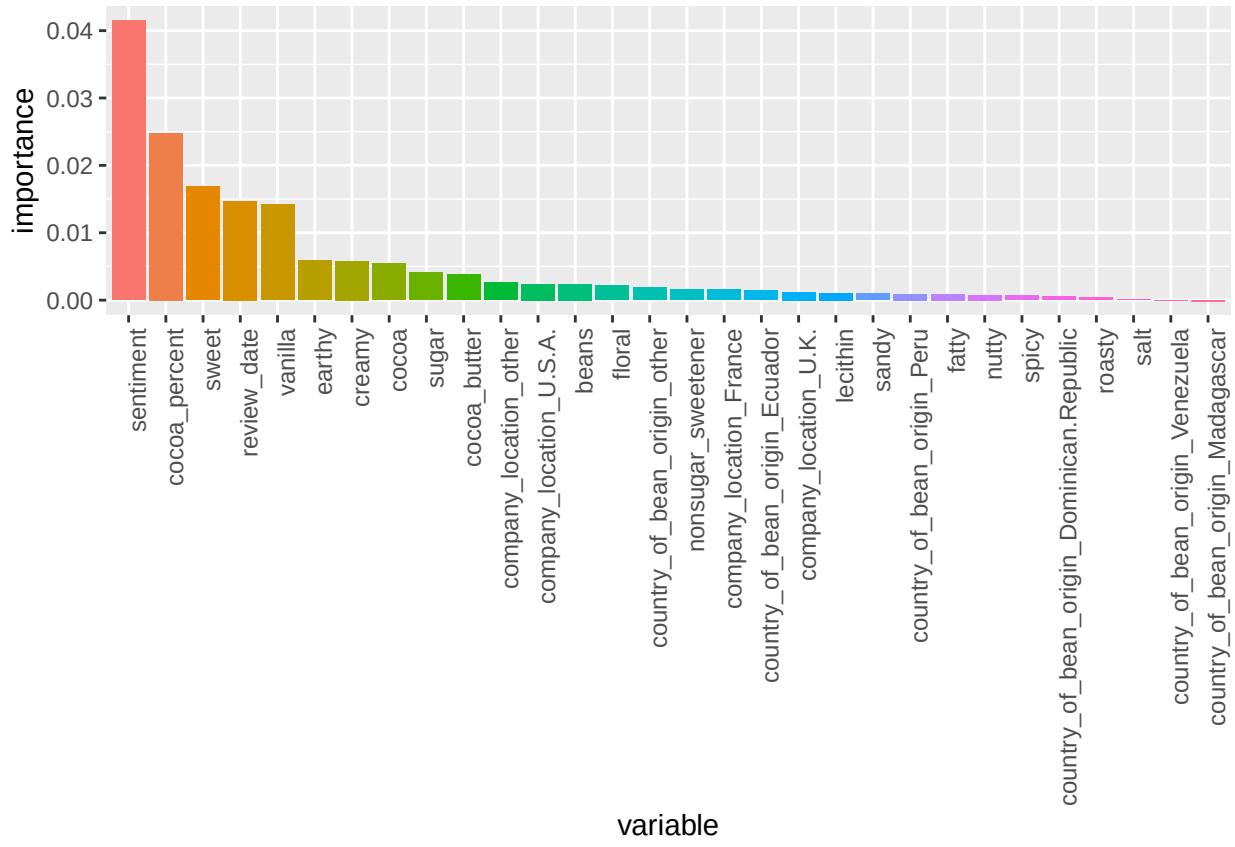
These predictions have a much higher range. MAE and RMSE are less than half what they were in regression

```
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

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.

```
rf_model |>
  extract_fit_engine() |>
  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))
```



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

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

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_er
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.

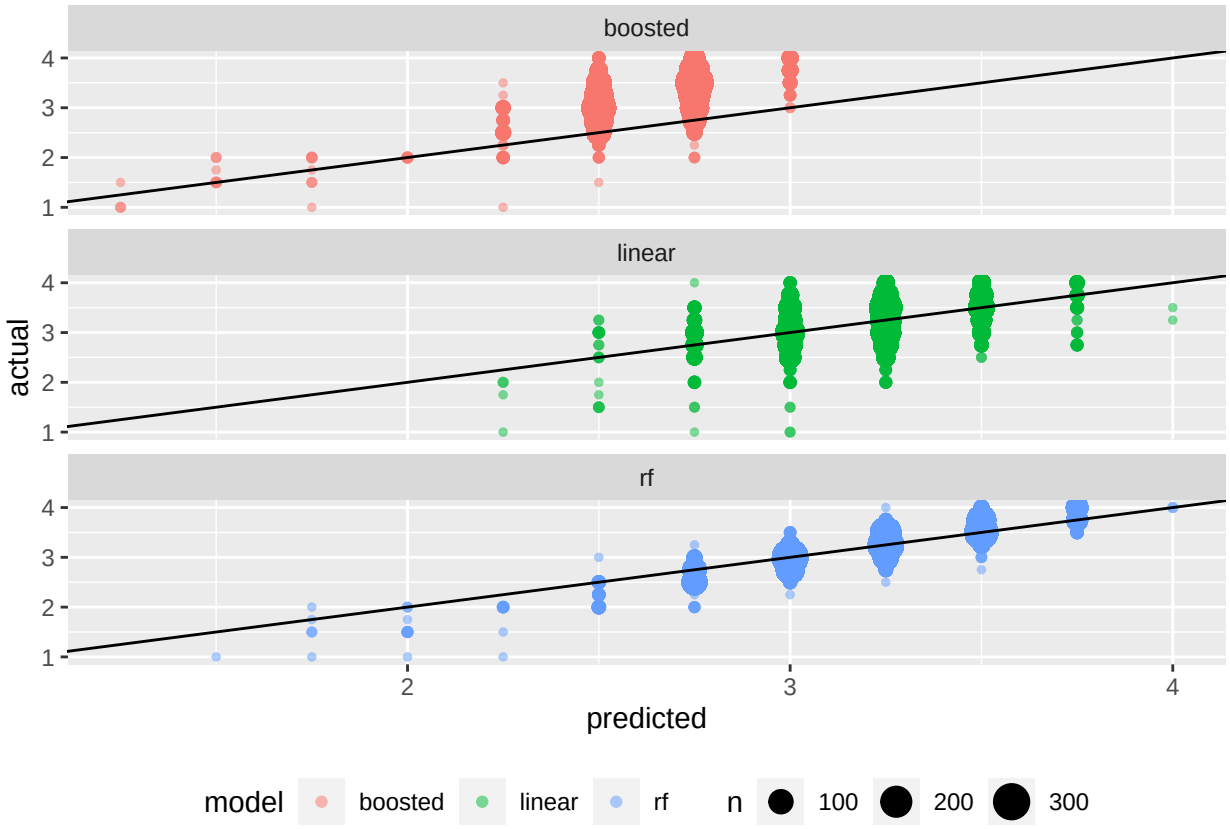
```
boost_preds <- augment(boost_model, new_data = train)
boost_preds[[".pred"]] <- fit_to_scale(boost_preds[[".pred"]], min = 1, max = 4, increment = .25)
chocolate_metrics(boost_preds, truth = rating, estimate = .pred)
```

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

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

Now, to 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.

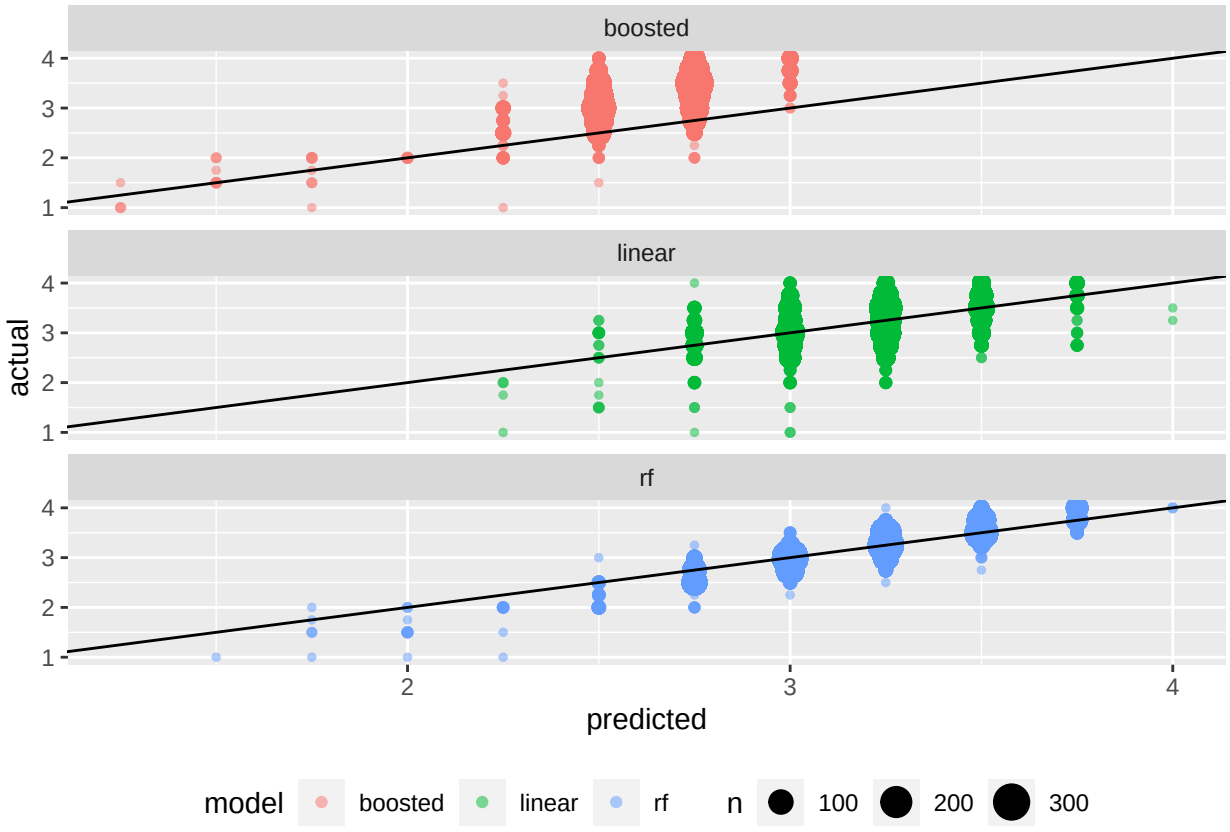
```
lin_test <- test_preds(lin_model, test)
rf_test <- test_preds(rf_model, test)
boost_test <- test_preds(boost_model, test)

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"
)
```

model	.metric	.estimator	.estimate
linear	mae	standard	0.3079662
linear	rmse	standard	0.3970624

model	.metric	.estimator	.estimate
linear	interval_error	standard	1.2211690
rf	mae	standard	0.3134230
rf	rmse	standard	0.4021690
rf	interval_error	standard	1.2527646
boost	mae	standard	0.5831447
boost	rmse	standard	0.6752469
boost	interval_error	standard	2.3270142

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.

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

Assessment

I enjoyed working on this project. I think I used an effective EDA strategy that identified ways the 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 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.