Regression Task - Abalone Age

Tolga Tabanli

2025-10-06

Intro

Abalones are sea snails that have an extremely strong shell with a unique structure. Aside from having been a food source for thousands of years, it is also a source of inspiration to material scientists. However, they comprise a critically endangered species. Although age can be used to determine its price and quality, it is also biologically important for use in scientific aquaculture, population and conservation studies. Their age can be reliably determined counting the rings under a microscope after cutting through shell, staining. A time-consuming procedure. How about predicting it using other physical measurements?

Setup

We first read the data:

```
library(tidyverse)
library(tidymodels)

abalone_raw <- read_csv("data/abalone.data",
                        col_names = c("sex", "length", "diameter", "height",
                                      "whole_weight", "shucked_weight", "viscera_weight",
                                      "shell_weight", "rings"))

# add 1.5 to ring number to get ages
abalone_raw$rings <- abalone_raw$rings + 1.5
```

Explore & visualize

What are the columns? What kind of data types are there? Any initial hypotheses? Any apparent factors for if the data might be unbalanced? Use `glimpse` to have a look.

```
# TODO
```

First let's see the data distributions for target and predictors. Use `pivot_longer` to collect variables under `variable` column with their values under `value` column. Plot **faceted** histograms using `ggplot2`. Stratify by sex to see if the data is unbalanced. this way, one can also see if the variable used for strata has any effect.
Hint: Use `geom_histogram()` with setting `fill` in the `aes` mapping to sex. Use `facet_wrap` with `scales = "free"` for facetting according to variables.

```
# TODO
```

Transformations

Optional: Try out some transformations to reduce skewness of data.
Hint: You can use `mutate()` with `across()` to specify the columns and the function to transform.
Example: `data %>% mutate(across(YOUR_COLUMNS), function)`
or with a formula: `data %>% mutate(across(YOUR_COLUMNS), ~ .x^2)`

```
# The variables `height`, `rings`, `shell_weight`, `shucked_weight`,
# `viscera_weight` and `whole_weight` seem right-skewed, while `diameter` and `length`
# are left-skewed. How can we transform these value so that they become relatively normal?

# TODO
```

Multicollinearity

Now that we adjusted and decided on our transformations, next step is checking collinearity. Select numeric columns (exclude id), calculate correlation matrix by `cor()` and plot using `corrplot()`.

```
# TODO
```

In case of high collinearity, there are a couple of things one could apply, such as correlation filtering, PCA or opting for tree-based models etc. We are going to use this problem to explore and compare different models and effect of some hyperparameters using `tidymodels` framework.

Modelling and training

Data split

Create the data split with 80 percent of data going. **Hint:** Use `initial_split()` to split data by setting 30 % of the observation aside for testing. Use the seed `2025` before the split for reproducibility.

```
# TODO
```

Tree-based models are known for their "immunity" against the collinearity problem. Additionally, regularized regression models, such as ridge and lasso, are also good for increasing stability and interpretability of ordinary regression under highly correlated predictors and low sample sizes. `penalty` parameter allows to adjust the level of shrinkage (lambda) and mixture controls how these two models are used simultaneously (`mixture = 0` is ridge, `mixture = 1` is lasso).

Here are some general considerations for preprocessing (not all might always apply):

- handle missingness
- handle skewness depending on prior exploration steps above. (You might need to check if there are negatives or zeros)
- center & scale
- eliminate near-zero variance features
- eliminate extremely correlated variables
- encode categorical variables as dummy or one-hot

For tree-based models, we do not need the steps for transformation and normalization. Similarly, we are not concerned about the collinearity.

Your Task

We define five subtasks:

1. Basic linear regression, no preprocessing
2. Basic linear regression with transformation
3. Basic linear regression with transformation and PCA, tune for num_comps
4. Elastic net regression, tune for penalty and mixture
5. A random forest, tune for feature selection (mtry), number of trees (trees) and node size (min_n). Use importance by impurity.
6. Specify the three recipes mentioned:
    a. A base recipe: ignore sex by using `update_role`, eliminate near zero variance features by `step_nzv`.
    b. Create new recipe based on a: Perform transformations with `step_sqrt()` and `step_mutate()`. Finally, `step_normalize()` on all numeric predictors.
    c. Create new recipe based on b: Apply PCA on all numeric predictors with `step_pca()`. Set the number of components to tune.
7. Specify the 3 models:
    a. linear_reg()
    b. linear_reg(penalty = tune(), mixture = tune()) with "glmnet" engine.
    c. rand_forest() with "ranger" engine and set importance to "impurity".
8. Create all 5 workflows by combining recipes and models, corresponding to the 5 subtasks.
9. We have some models requiring hyperparameter tuning (subtasks 3-5). Create hyperparameter grids for them using `grid_regular()`.

    Example:

```
grid <- grid_regular(mtry(range = c(2,8)),
                     trees(range = c(5, 70)),
                     levels = c(4, 5))
```

5. Perform hyperparameter tuning with grids using cross-validation using `vfold_cv()` and `tune_grid()`. Example:

```
cv_folds <- vfold_cv(train, v = 10)
tuned_rf <- tune_grid(YOUR_WORKFLOW_OBJECT,
                          resamples = cv_folds,
                          grid = grid_rf,
                          metrics = metric_set(mae, rmse, rsq))
```

6. Compare how hyperparameters affected performance in CV (**hint**: Use `autoplot()` on tuned objects.).

Here you find a template to help you in the process:

```
# === Recipes ===
# a. Base recipe
# TODO

# b. Transformation recipe
# TODO

# c. PCA recipe
# TODO
# ===============


# === Models ===
# basic linear regression
# TODO

# Elastic net regression, tune for penalty and mixture
# TODO

# Random forests
# TODO
# # ===============


# === Workflows ===
# 1. Basic reg, no preprocess
# TODO

# 2. Basic reg with transform preprocess
# TODO

# 3. Basic reg with PCA
# TODO

# 4. Elastic net
# TODO

# 5. Random forests
# TODO
# ===============


# Hyperparameter grid search for 3-6
# TODO

# === Cross-Validation ===
# TODO
# ========================

# Plot tuning results using autoplot() here:
# TODO
```

Finalizing workflow

Now that we have compared different models and tuned hyperparameters, we can select the best models where we did hyperparameter optimization and finalize workflow. These are done using exactly the same verbs in tidymodels as well! `select_best()` gives us the configuration of model parameters that have yielded the best performance. `finalize_workflow()` basically fills the gaps where we had set `tune()` with these best performing model settings.

```
# Select best model for each workflow
# TODO

# Finalize the models
# TODO
```

Lastly, fit all the finalized workflows on the data_split using the metrics mae, rmse and rsq. Save the results in a named list for easy manipulation later.

`Example:`

```
last_fits <- list(
  fit_basic_reg = last_fit(wf_basic_reg, split = data_split,
                           metrics = metric_set(mae, rmse, rsq)),
  fit_basic_reg_prep = last_fit(wf_basic_reg_prep, split = data_split,
                                metrics = metric_set(mae, rmse, rsq)),
  ...)
```

```
# last fits with all training data
# TODO
```

Here is a ready-to-use code snippet to plot the metric comparison assuming you named your list of fit models as `last_fits`.

```
# Plot MAE metric
mae_results <- map(last_fits, collect_metrics) %>%
  list_rbind(names_to = "model")

mae_results %>%
  ggplot(aes(x = model, y = .estimate)) +
  geom_col() +
  facet_wrap(~ .metric, scales = "free", ncol = 2) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust=1))
```

Variable importance can be shown using random forest's internal mechanisms. We had used `importance = "impurity"` for this purpose. This is a model-based importance.

Extract the fitted random forest model from your list by using `extract_fit_parsnip`. Use `vip()` from package `vip` on the fitted model to plot the importances.

```
# we have already computed impurity in original model
# we just need to extract the parsnip model

# TODO
```