Answering some {tidymodels} questions

After running the 'Introduction to machine learning with {tidymodels}' workshop as part of the R/Pharma 2023 Conference, there were a few questions that we didn't get the chance to answer. This blog post aims to answer some of them.

October 23, 2023

Last week I had the pleasure of running the *Introduction to machine learning with {tidymodels}* workshop as part of the R/Pharma 2023 Conference. Thanks again to Phil Bowsher for the invitation, Eric Nantz for TA'ing the workshop and keeping an eye on all the questions coming in, and Libby Heeren for so many helpful responses to questions!

During the workshop, we covered the pre-modelling steps with {tidymodels}, LASSO logistic regression, random forests, and support vector machines. The workshop materials can be found online:

- Slides: nrennie.github.io/r-pharma-2023-tidymodels
- GitHub: github.com/nrennie/r-pharma-2023-tidymodels

There were a few questions that we didn't get the chance to answer. And in some cases, now that I've had a bit more time to think, I have better answers for some of the questions I did answer! So I've shared those answers here instead.

Pre-processing

All the binary (explanatory) variables are defined as numeric 0s and 1s `<dbl>`, would it be better to convert them to factors?

We don't need to convert them to factors here, and actually you'll get an error if you do.

Is there a good standard range of proportions to use for the train vs test split of datasets?

This is one of those questions with no single answer. The `initial_split()` default is 75% training data. 80% training data is also commonly given as a starting point. With less training data, the parameter estimates have a higher variance. With less testing data, the performance measures have a higher variance. It also depends on how many observations you have - there is a tendency to use smaller testing sets for data that is very large. In contrast, for small datasets you may use something closer to a 50:50 split.

Does the `initial_split()` function account for class imbalance and maintain that proportion in the split?

To make sure that you have equivalent proportions of a particular variable in the resampled data as in the original data set, you can use the `strata` argument in the `initial_split()` function. By default, the `strata` argument is `NULL`, but you can provide a column name e.g. `hf_split <- initial_split(heart_failure, strata = death)` to conduct stratified sampling. Remember that when you perform the initial split, you haven't created `recipe` to define what the response variable is yet. There's a similar argument in `vfold_cv()` for cross-validation samples.

When you pre-process train and test, do you feed the parameters from train to process test. For example, when you normalise a variable in train does the normalised test have

mean and standard deviation different from 0?

The `step_normalize()` function creates a specification of a recipe step that will normalise numeric data to have a standard deviation of one and a mean of zero. `step_normalize()` estimates the variable standard deviations and means from the data used in the training argument. If you're applying the process to new data, you can use the `bake()` function to apply the scaling to the new data using these estimates.

More generally if you have a large enough sample size with a large imbalance in the outcome, would you recommend sub-sampling the larger class?

It depends! Most statistical models and machine learning algorithms are based on predicting the *average*. If you want to find out what factors affect the average observation, then sub-sampling is probably not helpful. If you want to make sure you accurately predict minority classes, then sub-sampling may be beneficial.

This blog post from Matt Kaye provides a nice discussion: matthewrkaye.com/posts/2023-03-25-balancing-classes/balancing-classes.htm.

In the scenario where you don't want to do a cross validation split, is it possible to split your data into train, validation and test splits?

Yes, you can use the `initial_validation_split()` instead of `initial_split()`. See here for more details: rsample.tidymodels.org/reference/initial_validation_split.html.

Is there a component to add a sensitive analysis in the recipe (i.e. I want independent covariates A:E), then perform the whole workflow again to then add covariate F too?

Yes, there are a set of functions in {workflows} that help you to add or remove variables from workflows. Look at the help functions for `workflow_variables()` by running `?workflow_variables()` to see examples of how it works.

How can I get a dataframe after the `step_dummy()` function?

To see what the `step_*` functions have done to the data, you can use `prep()` and `bake()` on the training data. For example:

Copy

```
# make the recipe as in the examples
hf_recipe <- recipe(death ~ ., data = hf_train) |>
  step_dummy(sex) |>
  step_normalize(age, serum_creatinine:time)

# return a tibble of the transformed training data
hf_recipe |>
  prep(training = hf_train) |>
  bake(new_data = NULL)
```

LASSO logistic regression

Is it necessary to add `family = "binary"` in `logistic_reg()`?

No, the `logistic_reg()` is specifically created for generalised linear models with binary outcomes - effectively the `family` argument is already defined. Unlike the `glm()` function in R which is for implementing multiple different types of generalised linear model - here, we need to be more specific and specify `family = "binary"`.

Why do you use `"glmnet"` in `set_engine()` for LASSO logistic regression, rather than another one from the list?

You can see all of the available engines using `show_engines()`. For example, for logistic regression we get:

Copy

```
show_engines("logistic_reg")
# A tibble: 7 × 2
  engine    mode
  <chr>     <chr>
1 glm       classification
2 glmnet    classification
3 LiblineaR classification
4 spark     classification
5 keras     classification
6 stan      classification
7 brulee    classification
```

The `logistic_reg()` function can actually fit different types of models including simple logistic regression models, LASSO regression models, ridge regression models, and a mixture of LASSO and ridge models. This means that there are multiple engines that fit each of these type of models - `glmnet` is used for LASSO regression.

The `logistic_reg()` function has an `engine` parameter - does it matter whether the engine is set within the `logistic_reg()` or outside using `set_engine()`?

No, you can use either. If you use both, it will use the last engine you specify last.

What are the advantages and disadvantages of LASSO logistic regression opposed to 'regular' logistic regression?

Advantages: the main advantage, in my experience, has been automatic variable selection - no need to use stepwise procedures!. This is especially useful if you have a high number of candidate covariates. By pushing coefficients towards zero, LASSO regression also helps to avoid over-fitting - making the model more likely to perform better on unseen data.

Disadvantages: By shrinking the coefficients towards zero, this means that the coefficients from a LASSO model don't represent the true relationship between an explanatory variable and the response. This makes models harder to interpret, and it's difficult to estimate uncertainty in these coefficients as they are biased towards zero. The introduction of the hyperparameter also adds an additional layer of complexity to fitting the model. LASSO regression doesn't usually perform well if you have lots of correlated variables - try using a ridge regression (or mixture) model instead.

Could you use LASSO to select the variables and then use those in a standard regression?

If your main aim is inference rather than prediction, then you're right that LASSO is likely not the most appropriate final model. Instead you are best to perform feature selection and feed the selected features into something like a standard regression model. LASSO is one method of feature selection, so yes you could. You could of course also make use of a different feature selection method.

Would using an Elastic Net (a trade off between LASSO and Ridge Regression) at this stage be appropriate?

Yes, it could definitely work, and it's actually a very small code change to implement. In the `mixture` argument of `logistic_reg()`, for LASSO regression we specify `mixture = 1`. If we specify `mixture = 0`, this implements a ridge regression model. Any value in between 0 and 1, specifies an elastic net model which mixes LASSO and ridge regression.

How did you take care of over-fitting the model?

LASSO regression penalises the model coefficients, and therefore is itself a way of avoiding over-fitting models. It shrinks the coefficients of some explanatory variables towards zero and helps you to avoid including additional variables that contribute towards over-fitting.

Is there variability around importance - similar to a confidence interval in OLS regression?

There are ways to measure the variability around the variable importance scores. For example, the {vip} package implements a permutation-based method. You can use this if you specify `method = "permute"` in the `vip::vi()` function. The permutation method allows you to compute standard errors (and other metrics) for the estimated variable importance scores. However, this isn't similar to confidence intervals for coefficients in OLS regression.

Read the {vip} package vignette for more details and examples: koalaverse.github.io/vip/articles/vip.html#permutation-method.

Model fitting in {tidymodels}

For each value of the hyperparameter, is its performance evaluated using CV? Or does each CV iteration use a different hyperparameter value?

The `vfold_cv()` has two arguments that control the number of times it is fitted. The `v` argument controls how many folds, the default is 10. The `repeats` argument controls how many times the v-fold partitioning is done. The default is 1.

For each cross-validation fold, the model is fitted to each element in the grid of penalty values. So if you have 10 cross validation folds, with 1 repeat, and 25 values of the hyperparameter, you'll have 250 fitted models in the tuning step. Try looking at the output `lasso_grid$.metrics`.

Can you elaborate what `extract_fit_parsnip()` does exactly ?

After you fit a model e.g. using the `fit()` function, you can extract the fitted parsnip model object. The parsnip object wraps the underlying model object e.g. the model object returned by `glmnet`. The output of `extract_fit_parsnip()` isn't usually very pretty so I'd recommend using `tidy()` to convert the output into a nice tibble. For example, from the LASSO fit in `example_01.R`:

Copy

```
final_lasso |>
  fit(hf_train) |>
  extract_fit_parsnip() |>
  tidy()

# A tibble: 12 × 3
   term                     estimate penalty
   <chr>                       <dbl>   <dbl>
 1 (Intercept)               -0.985   0.0596
 2 age                        0.0836  0.0596
 3 smoking                    0       0.0596
 4 anaemia                    0       0.0596
 5 diabetes                   0       0.0596
 6 high_blood_pressure        0       0.0596
 7 serum_creatinine           0.336   0.0596
 8 creatinine_phosphokinase   0       0.0596
 9 platelets                  0       0.0596
10 ejection_fraction         -0.270   0.0596
11 time                      -0.908   0.0596
12 sex_M                      0       0.0596
```

If the model has already been fitted during the tuning step can we not just extract it directly rather than fitting it again with `finalize_workflow()`?

Not in the examples here, although the model has been fitted many times during the tuning process, we never fit it to the whole training set during the tuning process - only to each of the cross-validation folds which are each a subset of the training data. We still need to calculate the rest of (non-tuned) parameters e.g. the LASSO coefficients when the model is fitted to the training data.

So, now that the model is ready, how do we use it to make predictions for new data?

As an example, let's assume we're looking at the LASSO model and you've run all the code in `example_01.R` and `example_02.R`. Let's also assume that you have some `new_data` with the same format and columns as the original data, but without the response variable column. You can use the `predict()` function to apply the model to new data:

Copy

```
final_lasso |>
  fit(hf_train) |>
  predict(new_data)
```

Is `last_fit()` taking care of applying pre-processing steps to testing set?

Yes, this is the first (and only) time we're actually using the testing data.

Different types of models

Can this be used for deep learning as well?

You can fit neural networks with {tidymodels}. There's some nice examples in the {tidymodels} documentation here: https://www.tidymodels.org/learn/models/parsnip-nnet/. If you're interested in deep learning in R, I'd also recommend having a look at *Deep Learning and Scientific Computing with R `torch`*. It uses the R interface to PyTorch, for an alternative approach to deep learning in R.

For models fitted in Stan are these precompiled?

I believe so. The Stan models in {tidymodels} using packages such as `{rstanarm}` to actually fit the models. Check the details of specific engines for more information.

What about XGBoost?

Yes, you can fit `XGBoost` models with {tidymodels}. Julia Silge has a nice blog post on using XGBoost to predict beach volleyball matches: juliasilge.com/blog/xgboost-tune-volleyball/

Other questions

Can you show how to read a confusion matrix?

Yes, let's say that the output of `conf_mat()` in `example_02.R` looks like this:

Copy

```
            Truth
Prediction  0  1
         0 37  9
         1  1 13
```

This means that: there were 37 zeros that we correctly predicted were zeros; and there was 1 zero that we incorrectly labelled as a one. There were 9 ones that we incorrectly predicted were zeros; and there were 13 ones that we correctly predicted were ones. Ideally only the diagonal line from the top-left to the bottom-right will have non-zero values.

Should I be worried that I got slightly different values even when I followed the same steps during live coding and used `set.seed()`?

I wouldn't worry about it too much - it's very possible I accidentally ran something twice in the workshop!

I'm wondering whether using {tidymodels} has any advantages/disadvantages compared to using scikit-learn?

The main benefit of {tidymodels} is that it bring machine learning capabilities to R in a very accessible, user-friendly way. This means that if you're already doing exploratory data analysis, data preparation, and results visualisation in R you can continue doing so without having to export data and run models in Python instead. Multi-language workflows can be great, but single-language workflows tend to be easier to maintain and debug.

I don't think I would ever say one is better than the other - I'd choose the between them based on the language I (or the team) are already working in.

What would you recommend as kind of the next step in learning to someone who is new to this?

Any of the following resources are fantastic next steps:

- {tidymodels} documentation.
- Tidy Modeling with R. You can also join the Tidy Modelling with R book club in the R4DS Online Learning Community Slack channel.

- Blog by Julia Silge.

I'm hoping this has answered most of the remaining questions!

Image: giphy.com

For attribution, please cite this work as:

> **Answering some {tidymodels} questions**.
> Nicola Rennie. October 23, 2023.
> nrennie.rbind.io/blog/answering-some-tidymodels-questions

1 Comment - *powered by utteranc.es*

durraniu commented on 23. Okt. 2023

Thank you for teaching the workshop and following up with the questions.

👍 1

Write Preview

Sign in to comment

Styling with Markdown is supported Sign in with GitHub

← Creating typewriter-styled maps in {ggplot2} Making art in Python with plotnine →
