Some basics before we go on:  
Normally I would do an extensive exploration of the data to see how the variables  
interact and influence the house price. In this example I want to focus on  
how you can use lightgbm with tidymodels, so I skip this part and use Andy and  
Nick’s feature engineering with a small change.

**Basic steps for machine learning projects**

The steps in most machine learning projects are as follows:

* Loading necessary packages and data
* split data into train and test ({rsample})
* light preprocessing ({recipes})
* find the best hyperparameters by
  + creating crossvalidation folds ({rsample})
  + creating a model specification ({tune, parsnip, treesnip, dials})
  + creating a grid of values ({dials})
  + using a workflow to contain the model and formula ({workflows})
  + tune the model ({tune})
* find the best model from tuning
* retrain on entire test data
* evaluate on test data ({yardstick})
* check residuals and model diagnostics

**Loading necessary packages and data**

*# data*

**library**(AmesHousing)

*# data cleaning*

**library**(janitor)

*# data prep*

**library**(dplyr)

*# visualisation*

**library**(ggplot2)

*# tidymodels*

**library**(rsample)

**library**(recipes)

**library**(parsnip)

**library**(tune)

**library**(dials)

**library**(workflows)

**library**(yardstick)

**library**(treesnip)

Setting up some settings, this is optional but can help speed things  
up.

*##*

*# speed up computation with parallel processing*

**library**(doParallel)

all\_cores **<-** parallel**::detectCores**(logical **=** **FALSE**)

**registerDoParallel**(cores **=** all\_cores)

And setting up data:

*# set the random seed so we can reproduce any simulated results.*

**set.seed**(1234)

*# load the housing data and clean names*

ames\_data **<-** **make\_ames**() **%>%**

janitor**::clean\_names**()

**Split data into train and test**

ames\_split **<-** rsample**::initial\_split**(

ames\_data,

prop **=** 0.8,

strata **=** sale\_price

)

**Some light preprocessing**

Many models require careful and extensive variable preprocessing to produce  
accurate predictions.  
Boosted tree models like XGBoost,lightgbm, and catboost are quite robust against  
highly skewed and/or correlated data, so the amount of preprocessing required  
is minimal. In contrast to XGBoost, both lightgbm and catboost are very capable  
of handling categorical variables (factors) and so you don’t need to turn  
variables into dummies (one hot encode), in fact you shouldn’t do it, it makes  
everything slower and might give you worse performance.

preprocessing\_recipe **<-**

recipes**::recipe**(sale\_price **~** ., data **=** **training**(ames\_split)) **%>%**

*# combine low frequency factor levels*

recipes**::step\_other**(**all\_nominal**(), threshold **=** 0.01) **%>%**

*# remove no variance predictors which provide no predictive information*

recipes**::step\_nzv**(**all\_nominal**()) **%>%**

*# prep the recipe so it can be used on other data*

**prep**()

**Find the best hyperparameters**

Create crossvalidation folds. This means the data is split into 5 chunks, the model trained on four of them and predicts on the fifth chunk. This  
is done five times (predicting every time on a different chunk) and the  
metrics will be averaged over the chunks as a measure of out of sample performance.

ames\_cv\_folds **<-**

recipes**::bake**(

preprocessing\_recipe,

new\_data **=** **training**(ames\_split)

) **%>%**

rsample**::vfold\_cv**(v **=** 5)

Create a model specification for lightgbm  
The treesnip package makes sure that boost\_tree understands what engine  
lightgbm is, and how the parameters are translated internaly.  
We don’t know yet what the ideal parameter values are for  
this lightgbm model. So we have to tune the parameters.

lightgbm\_model**<-**

parsnip**::boost\_tree**(

mode **=** "regression",

trees **=** 1000,

min\_n **=** **tune**(),

tree\_depth **=** **tune**(),

) **%>%**

**set\_engine**("lightgbm", objective **=** "reg:squarederror",verbose**=**-1)

Grid specification by dials package to fill in the model above  
This specification automates the min and max values of these  
parameters.

According to the [lightgbm parameter tuning guide](https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html)  
the hyperparameters number of leaves, min\_data\_in\_leaf, and max\_depth are the most important features.  
Currently implemented features for lightgbm are:

* feature\_fraction (mtry)
* num\_iterations (trees)
* min\_data\_in\_leaf (min\_n)
* max\_depth (tree\_depth)
* learning\_rate (learn\_rate)
* min\_gain\_to\_split (loss\_reduction)
* bagging\_fraction (sample\_size)

(so we do not yet have number of leaves).

lightgbm\_params **<-**

dials**::parameters**(

*# The parameters have sane defaults, but if you have some knowledge*

*# of the process you can set upper and lower limits to these parameters.*

**min\_n**(), *# 2nd important*

**tree\_depth**() *# 3rd most important*

)

And finally construct a grid with actual values to search for.

lgbm\_grid **<-**

dials**::grid\_max\_entropy**(

lightgbm\_params,

size **=** 30 *# set this to a higher number to get better results*

*# I don't want to run this all night, so I set it to 30*

)

**head**(lgbm\_grid)

# A tibble: 6 x 2

min\_n tree\_depth

1 10 9

2 19 11

3 40 8

4 12 1

5 19 5

6 31 8

To tune our model, we perform grid search over our lightgbm\_grid’s grid space  
to identify the hyperparameter values that have the lowest prediction error.

Actual workflow object

lgbm\_wf **<-**

workflows**::workflow**() **%>%**

**add\_model**(lightgbm\_model

) **%>%**

**add\_formula**(sale\_price **~** .)

**so far little to no computation has been performed except for preprocessing calculations**  
But the machine will start to run hot in the next step, where we call tune\_grid.  
If you look at the process for xgboost and in the next tutorial for catboost  
the steps remain the same, with a few details different but mostly the same!

We call tune\_grid with:

* “object”: lgbm\_wf which is a workflow that we defined by the parsnip and  
  workflows packages
* “resamples”: ames\_cv\_folds as defined by rsample and  
  recipes packages
* “grid”: lgbm\_grid our grid space as defined by the dials  
  package
* “metric”: the yardstick package defines the metric set used to  
  evaluate model performance

lgbm\_tuned **<-** tune**::tune\_grid**(

object **=** lgbm\_wf,

resamples **=** ames\_cv\_folds,

grid **=** lgbm\_grid,

metrics **=** yardstick**::metric\_set**(rmse, rsq, mae),

control **=** tune**::control\_grid**(verbose **=** **FALSE**) *# set this to TRUE to see*

*# in what step of the process you are. But that doesn't look that well in*

*# a blog.*

)

**Find the best model from tuning results**

hyperparameter values which performed best at minimizing RMSE.

lgbm\_tuned **%>%**

tune**::show\_best**(metric **=** "rmse",n **=** 5)

# A tibble: 5 x 8

min\_n tree\_depth .metric .estimator mean n std\_err .config

1 8 3 rmse standard 24555. 5 1192. Model30

2 13 4 rmse standard 24647. 5 1042. Model29

3 12 1 rmse standard 25195. 5 1281. Model04

4 19 5 rmse standard 25206. 5 1079. Model05

5 6 5 rmse standard 25382. 5 858. Model18

plot the performance per parameter.

lgbm\_tuned **%>%**

tune**::show\_best**(metric **=** "rmse",n **=** 10) **%>%**

tidyr**::pivot\_longer**(min\_n**:**tree\_depth, names\_to**=**"variable",values\_to**=**"value" ) **%>%**

**ggplot**(**aes**(value,mean)) **+**

**geom\_line**(alpha**=**1**/**2)**+**

**geom\_point**()**+**

**facet\_wrap**(**~**variable,scales **=** "free")**+**

**ggtitle**("Best parameters for RMSE")

Since we asked for multiple metrics we can see the best performance for different  
metrics too.

lgbm\_tuned **%>%**

tune**::show\_best**(metric **=** "mae",n **=** 10) **%>%**

tidyr**::pivot\_longer**(min\_n**:**tree\_depth, names\_to**=**"variable",values\_to**=**"value" ) **%>%**

**ggplot**(**aes**(value,mean)) **+**

**geom\_line**(alpha**=**1**/**2)**+**

**geom\_point**()**+**

**facet\_wrap**(**~**variable,scales **=** "free")**+**

**ggtitle**("Best parameters for MAE")

Than we can select the best parameter combination for a metric, or do it manually.

lgbm\_best\_params **<-**

lgbm\_tuned **%>%**

tune**::select\_best**("rmse")

Finalize the lgbm model to use the best tuning parameters.

lgbm\_model\_final **<-**

lightgbm\_model**%>%**

**finalize\_model**(lgbm\_best\_params)

The finalized model is filled in:

*# empty*

lightgbm\_model

Boosted Tree Model Specification (regression)

Main Arguments:

trees = 1000

min\_n = tune()

tree\_depth = tune()

Engine-Specific Arguments:

objective = reg:squarederror

verbose = -1

Computational engine: lightgbm

*# filled in*

lgbm\_model\_final

Boosted Tree Model Specification (regression)

Main Arguments:

trees = 1000

min\_n = 8

tree\_depth = 3

Engine-Specific Arguments:

objective = reg:squarederror

verbose = -1

Computational engine: lightgbm

**Retrain on entire training data**

This is giving me a lot of warnings and lgbm isn’t giving us the warnings in warning type, so I’ve hidden this ugly part. **click to unhide** But I fit the finalized model again on new data.

**And evaluate on test data (yardstick)**

test\_processed **<-** **bake**(preprocessing\_recipe, new\_data **=** **testing**(ames\_split))

test\_prediction **<-**

trained\_model\_all\_data **%>%**

*# use the training model fit to predict the test data*

**predict**(new\_data **=** test\_processed) **%>%**

**bind\_cols**(**testing**(ames\_split))

measure the accuracy of our model on training set (overestimation)

train\_prediction **%>%**

yardstick**::metrics**(sale\_price, .pred) **%>%**

**mutate**(.estimate **=** **format**(**round**(.estimate, 2), big.mark **=** ",")) **%>%**

knitr**::kable**()

| **.metric** | **.estimator** | **.estimate** |
| --- | --- | --- |
| rmse | standard | 17,575.38 |
| rsq | standard | 0.95 |
| mae | standard | 12,473.87 |

measure the accuracy of our model on data it hasn’t seen before (testset)

test\_prediction **%>%**

yardstick**::metrics**(sale\_price, .pred) **%>%**

**mutate**(.estimate **=** **format**(**round**(.estimate, 2), big.mark **=** ",")) **%>%**

knitr**::kable**()

| **.metric** | **.estimator** | **.estimate** |
| --- | --- | --- |
| rmse | standard | 33,966.76 |
| rsq | standard | 0.83 |
| mae | standard | 18,040.51 |

Not a bad score.

**look at residuals**

house\_prediction\_residual **<-** test\_prediction **%>%**

**arrange**(.pred) **%>%**

**mutate**(residual\_pct **=** (sale\_price **-** .pred) **/** .pred) **%>%**

**select**(.pred, residual\_pct)

**ggplot**(house\_prediction\_residual, **aes**(x **=** .pred, y **=** residual\_pct)) **+**

**geom\_point**() **+**

**xlab**("Predicted Sale Price") **+**

**ylab**("Residual (%)") **+**

**scale\_x\_continuous**(labels **=** scales**::dollar\_format**()) **+**

**scale\_y\_continuous**(labels **=** scales**::**percent)

So that works quite well, there are some outliers in low price. And we can probably discover what cases are doing badly and maybe add more information to the model to discover and distinguish.

**Other stuff**

Installing lightgbm can be done following the [official instructions](https://lightgbm.readthedocs.io/en/latest/R/index.html)

**Reproducibility**

At the moment of creation (when I knitted this document ) this was the state of my machine: **click to expand**

sessioninfo**::session\_info**()

─ Session info ───────────────────────────────────────────────────────────────

setting value

version R version 4.0.2 (2020-06-22)

os macOS Catalina 10.15.6

system x86\_64, darwin17.0

ui X11

language (EN)

collate en\_US.UTF-8

ctype en\_US.UTF-8

tz Europe/Amsterdam

date 2020-08-27

─ Packages ───────────────────────────────────────────────────────────────────

package \* version date lib source

AmesHousing \* 0.0.4 2020-06-23 [1] CRAN (R 4.0.2)

assertthat 0.2.1 2019-03-21 [1] CRAN (R 4.0.0)

class 7.3-17 2020-04-26 [1] CRAN (R 4.0.2)

cli 2.0.2 2020-02-28 [1] CRAN (R 4.0.0)

codetools 0.2-16 2018-12-24 [1] CRAN (R 4.0.2)

colorspace 1.4-1 2019-03-18 [1] CRAN (R 4.0.0)

crayon 1.3.4 2017-09-16 [1] CRAN (R 4.0.0)

data.table 1.13.0 2020-07-24 [1] CRAN (R 4.0.2)

dials \* 0.0.8 2020-07-08 [1] CRAN (R 4.0.2)

DiceDesign 1.8-1 2019-07-31 [1] CRAN (R 4.0.0)

digest 0.6.25 2020-02-23 [1] CRAN (R 4.0.0)

doParallel \* 1.0.15 2019-08-02 [1] CRAN (R 4.0.2)

dplyr \* 1.0.2 2020-08-18 [1] CRAN (R 4.0.2)

ellipsis 0.3.1 2020-05-15 [1] CRAN (R 4.0.1)

evaluate 0.14 2019-05-28 [1] CRAN (R 4.0.0)

fansi 0.4.1 2020-01-08 [1] CRAN (R 4.0.0)

farver 2.0.3 2020-01-16 [1] CRAN (R 4.0.0)

foreach \* 1.5.0 2020-03-30 [1] CRAN (R 4.0.1)

furrr 0.1.0 2018-05-16 [1] CRAN (R 4.0.0)

future 1.18.0 2020-07-09 [1] CRAN (R 4.0.2)

generics 0.0.2 2018-11-29 [1] CRAN (R 4.0.0)

ggplot2 \* 3.3.2 2020-06-19 [1] CRAN (R 4.0.1)

globals 0.12.5 2019-12-07 [1] CRAN (R 4.0.0)

glue 1.4.1 2020-05-13 [1] CRAN (R 4.0.1)

gower 0.2.2 2020-06-23 [1] CRAN (R 4.0.1)

GPfit 1.0-8 2019-02-08 [1] CRAN (R 4.0.0)

gtable 0.3.0 2019-03-25 [1] CRAN (R 4.0.0)

highr 0.8 2019-03-20 [1] CRAN (R 4.0.0)

htmltools 0.5.0 2020-06-16 [1] CRAN (R 4.0.1)

ipred 0.9-9 2019-04-28 [1] CRAN (R 4.0.0)

iterators \* 1.0.12 2019-07-26 [1] CRAN (R 4.0.0)

janitor \* 2.0.1 2020-04-12 [1] CRAN (R 4.0.2)

jsonlite 1.7.0 2020-06-25 [1] CRAN (R 4.0.1)

knitr 1.29 2020-06-23 [1] CRAN (R 4.0.1)

labeling 0.3 2014-08-23 [1] CRAN (R 4.0.0)

lattice 0.20-41 2020-04-02 [1] CRAN (R 4.0.2)

lava 1.6.7 2020-03-05 [1] CRAN (R 4.0.0)

lhs 1.0.2 2020-04-13 [1] CRAN (R 4.0.2)

lifecycle 0.2.0 2020-03-06 [1] CRAN (R 4.0.0)

lightgbm 3.0.0-1 2020-08-26 [1] url

listenv 0.8.0 2019-12-05 [1] CRAN (R 4.0.0)

lubridate 1.7.9 2020-06-08 [1] CRAN (R 4.0.2)

magrittr 1.5 2014-11-22 [1] CRAN (R 4.0.0)

MASS 7.3-51.6 2020-04-26 [1] CRAN (R 4.0.2)

Matrix 1.2-18 2019-11-27 [1] CRAN (R 4.0.2)

munsell 0.5.0 2018-06-12 [1] CRAN (R 4.0.0)

nnet 7.3-14 2020-04-26 [1] CRAN (R 4.0.2)

parsnip \* 0.1.3 2020-08-04 [1] CRAN (R 4.0.2)

pillar 1.4.6 2020-07-10 [1] CRAN (R 4.0.2)

pkgconfig 2.0.3 2019-09-22 [1] CRAN (R 4.0.0)

plyr 1.8.6 2020-03-03 [1] CRAN (R 4.0.0)

pROC 1.16.2 2020-03-19 [1] CRAN (R 4.0.0)

prodlim 2019.11.13 2019-11-17 [1] CRAN (R 4.0.0)

purrr 0.3.4 2020-04-17 [1] CRAN (R 4.0.1)

R6 2.4.1 2019-11-12 [1] CRAN (R 4.0.0)

Rcpp 1.0.5 2020-07-06 [1] CRAN (R 4.0.2)

recipes \* 0.1.13 2020-06-23 [1] CRAN (R 4.0.1)

rlang 0.4.7 2020-07-09 [1] CRAN (R 4.0.2)

rmarkdown 2.3 2020-06-18 [1] CRAN (R 4.0.1)

rpart 4.1-15 2019-04-12 [1] CRAN (R 4.0.2)

rsample \* 0.0.7 2020-06-04 [1] CRAN (R 4.0.1)

scales \* 1.1.1 2020-05-11 [1] CRAN (R 4.0.1)

sessioninfo 1.1.1 2018-11-05 [1] CRAN (R 4.0.1)

snakecase 0.11.0 2019-05-25 [1] CRAN (R 4.0.2)

stringi 1.4.6 2020-02-17 [1] CRAN (R 4.0.0)

stringr 1.4.0 2019-02-10 [1] CRAN (R 4.0.0)

survival 3.2-3 2020-06-13 [1] CRAN (R 4.0.1)

tibble 3.0.3 2020-07-10 [1] CRAN (R 4.0.2)

tidyr 1.1.1 2020-07-31 [1] CRAN (R 4.0.2)

tidyselect 1.1.0 2020-05-11 [1] CRAN (R 4.0.1)

timeDate 3043.102 2018-02-21 [1] CRAN (R 4.0.0)

treesnip \* 0.1.0 2020-08-26 [1] Github (curso-r/treesnip@8a87e8c)

tune \* 0.1.1 2020-07-08 [1] CRAN (R 4.0.2)

utf8 1.1.4 2018-05-24 [1] CRAN (R 4.0.0)

vctrs 0.3.2 2020-07-15 [1] CRAN (R 4.0.2)

withr 2.2.0 2020-04-20 [1] CRAN (R 4.0.2)

workflows \* 0.1.2 2020-07-07 [1] CRAN (R 4.0.2)

xfun 0.15 2020-06-21 [1] CRAN (R 4.0.2)

yaml 2.2.1 2020-02-01 [1] CRAN (R 4.0.0)

yardstick \* 0.0.7 2020-07-13 [1] CRAN (R 4.0.2)

[1] /Library/Frameworks/R.framework/Versions/4.0/Resources/library