ML-Tools-Assignment-1

Gyongyver Kamenar (2103380)

3/14/2022

```
library(ranger)
library(tidyverse)
library(caret)
library(kableExtra)
library(lubridate)
library(ggpubr)

# Set my theme
devtools::source_url('https://raw.githubusercontent.com/gyongyver-droid/ceu-data-analysis/master/Assignm
theme_set(theme_gyongyver())
```

Problem 1

```
caravan_data <- as_tibble(ISLR::Caravan)

set.seed(20220310)
caravan_sample <- slice_sample(caravan_data, prop = 0.2)
n_obs <- nrow(caravan_sample)
test_share <- 0.2

test_indices <- sample(seq(n_obs), floor(test_share * n_obs))
caravan_test <- slice(caravan_sample, test_indices)
caravan_train <- slice(caravan_sample, -test_indices)
```

A)

What would be a good evaluation metric for this problem? Think about it from the perspective of the business.

A good metric is how accurately we can predict, that the caravan will purchase insurance. 100% accuracy means that we predict exactly the actual case, 0% certainty means that we predict the opposite (since it binary) in each cases. Accuracy can be calculated by adding the number of correctly classified items and divide it by the total number of items. We can also add a confidence interval using the standard deviation of the accuracy measure.

B) and C)

Let's use the basic metric for classification problems: the accuracy (% of correctly classified examples). Train a simple logistic regression (using all of the available variables) and evaluate its performance on both the train and the test set. Does this model perform well? (Hint: You might want to evaluate a basic benchmark model first for comparison – e.g. predicting that no one will make a purchase.) Let's say your accuracy is 95%. Do

we know anything about the remaining 5%? Why did we mistake them? Are they people who bought and we thought they won't? Or quite the opposite? Report a table about these mistakes (Hint: I would like to see the Confusion Matrix.)

```
mean(as.numeric(caravan_train$Purchase)) %>% kable() %>%kable_styling(position = "center",latex_options
```

x
1.059013

```
caravan_train %>% group_by(Purchase) %>% count() %>% kable() %>%kable_styling(position = "center",latex
```

Purchase	n
No	877
Yes	55

The average can be the basic benchmark model, we can see that the average of the training data is 1.0590129 (the labels are 1 and 2). This means, that if we predict 1 (No insurance) for all of them, we will mistake in just 5.9% of the cases. This really simple model already results 94.0987124 % accuracy. Applying this benchmark model on the test set, we get 94.8275862% accuracy.

```
set.seed(20220310)

vars2 <- names(caravan_train[,1:85])

form <- formula(paste0("Purchase ~", paste0(vars2, collapse = " + ")))

ctrl <- trainControl(method = "cv", savePredictions = "final", returnResamp = "final")

logit_model <- train(
    form = form,
    method     = "glm",
    data       = caravan_train,
    family     = binomial,
    trControl = ctrl
  )

logit_model$results %>% kable() %>% kable_styling(latex_options = "HOLD_position")# Accuracy
```

parameter	Accuracy	Kappa	AccuracySD	KappaSD
none	0.9238489	0.0165315	0.0169802	0.0918588

```
#logit_model$pred
```

I trained a logit model with cross-validation and with all variables, and it has 0.9238489 accuracy, which is quite bad compared to the benchmark model. So let's see the further details of the model and the predictions.

```
show_confmat_output<-function(cm, model_name="model"){
  print(model_name %>% kable(col.names = "") %>%kable_styling(position = "center",latex_options = "HOLD
  print(cm$table %>% kable()%>%kable_styling(position = "center",latex_options = "HOLD_position"))
  print(cm$overall %>% kable(col.names = model_name)%>%kable_styling(position = "center",latex_options =
  print(cm$byClass %>% kable(col.names = model_name)%>%kable_styling(position = "center",latex_options =
}
```

```r
# Train evaluation
cm_logit<-confusionMatrix(logit_model$pred$pred,logit_model$pred$obs)
show_confmat_output(cm_logit,"Logit model train")
```

Logit model train

	No	Yes
No	859	53
Yes	18	2

	Logit model train
Accuracy	0.9238197
Kappa	0.0225702
AccuracyLower	0.9048773
AccuracyUpper	0.9400262
AccuracyNull	0.9409871
AccuracyPValue	0.9866976
McnemarPValue	0.0000546

	Logit model train
Sensitivity	0.9794755
Specificity	0.0363636
Pos Pred Value	0.9418860
Neg Pred Value	0.1000000
Precision	0.9418860
Recall	0.9794755
F1	0.9603130
Prevalence	0.9409871
Detection Rate	0.9216738
Detection Prevalence	0.9785408
Balanced Accuracy	0.5079196

```r
cm_logit_test<-confusionMatrix(predict(logit_model,caravan_test),caravan_test$Purchase) # Test Accuracy

show_confmat_output(cm_logit_test,"Logit model test")
```

Logit model test

	No	Yes
No	216	12
Yes	4	0

	Logit model test
Accuracy	0.9310345
Kappa	-0.0265487
AccuracyLower	0.8904188
AccuracyUpper	0.9600692
AccuracyNull	0.9482759
AccuracyPValue	0.9044737
McnemarPValue	0.0801183

	Logit model test
Sensitivity	0.9818182
Specificity	0.0000000
Pos Pred Value	0.9473684
Neg Pred Value	0.0000000
Precision	0.9473684
Recall	0.9818182
F1	0.9642857
Prevalence	0.9482759
Detection Rate	0.9310345
Detection Prevalence	0.9827586
Balanced Accuracy	0.4909091

On the train dataset, the logit model classified 861 items correctly out of the 932. The logit predicted 18 'Yes' while actually these do not have insurence (FN) and predicted 53 'No' while actually these have/pay insurance (FP). We can see, that the number of false positive classification is much higher than the number of false negative. (Positive class is 'No') The sensitivity of the prediction is 0.9794755 while the specificity is 0.0363636. This means, that the model classified 3.6363636% of the negative cases correctly. The positive predictive value is 0.941886 which is not so bad, however, the negative predictive value is 0.1 which is really low. It basically means, that the model's negative classifications is in only 10% true.

On the test dataset, the accuracy is 0.9310345. The FN rate is 0.0172414 while the FP rate is 0.0517241. The sensitivity is 0.9818182 while the specificity is 0. Just like on the train set, the false positive rate is much higher then the false negative rate.

D)

What do you think is a more serious mistake in this situation?

The more serious mistake is, when we think/predict that a customer will purchase insurance but actually she/he does not (false negative case) . Because in this case, the insurance company will lose a lot of money they expected to have. On the other hand, if the model gives no insurance classification but actually the customer purchased an insurance, it is additional money they did not expect, which is overall not so bad until it's rate is not too high, so the company can handle these cases (e.g. enough employees) .

E)

You might have noticed (if you checked your data first) that most of your features are categorical variables coded as numbers. Turn your features into factors and rerun your logistic regression. Did the prediction performance improve?

```
caravan_data <-caravan_data %>% mutate_all(
  as.factor
)
```

```r
set.seed(20220310)
caravan_sample <- slice_sample(caravan_data, prop = 0.2)
n_obs <- nrow(caravan_sample)
test_share <- 0.2

test_indices <- sample(seq(n_obs), floor(test_share * n_obs))
caravan_test <- slice(caravan_sample, test_indices)
caravan_train <- slice(caravan_sample, -test_indices)

# rerun logit model with factors
logit_model_factor <- train(
    form = formula(paste0("Purchase ~", paste0(vars2, collapse = " + "))),
    method     = "glm",
    data       = caravan_train,
    family     = binomial,
     trControl = ctrl
  )

logit_model_factor$results %>% kable() %>% kable_styling(latex_options = "HOLD_position") # Accurary 0.
```

parameter	Accuracy	Kappa	AccuracySD	KappaSD
none	0.7780977	-0.0138879	0.0822033	0.0947437

```r
# Train evaluation
show_confmat_output(confusionMatrix(logit_model_factor$pred$pred,logit_model_factor$pred$obs), "Logit w
```

Logit with factors - train

	No	Yes
No	717	47
Yes	160	8

	Logit with factors - train
Accuracy	0.7778970
Kappa	-0.0188428
AccuracyLower	0.7498184
AccuracyUpper	0.8042068
AccuracyNull	0.9409871
AccuracyPValue	1.0000000
McnemarPValue	0.0000000

	Logit with factors - train
Sensitivity	0.8175599
Specificity	0.1454545
Pos Pred Value	0.9384817
Neg Pred Value	0.0476190
Precision	0.9384817
Recall	0.8175599
F1	0.8738574
Prevalence	0.9409871
Detection Rate	0.7693133
Detection Prevalence	0.8197425
Balanced Accuracy	0.4815072

```
# Test evaluation

show_confmat_output(confusionMatrix(predict(logit_model_factor,caravan_test),caravan_test$Purchase), "L
```

Logit with factors - test

	No	Yes
No	194	8
Yes	26	4

	Logit with factors - test
Accuracy	0.8534483
Kappa	0.1258865
AccuracyLower	0.8012741
AccuracyUpper	0.8963251
AccuracyNull	0.9482759
AccuracyPValue	1.0000000
McnemarPValue	0.0035515

	Logit with factors - test
Sensitivity	0.8818182
Specificity	0.3333333
Pos Pred Value	0.9603960
Neg Pred Value	0.1333333
Precision	0.9603960
Recall	0.8818182
F1	0.9194313
Prevalence	0.9482759
Detection Rate	0.8362069
Detection Prevalence	0.8706897
Balanced Accuracy	0.6075758

The performance of the model did not improve as we can see from the reports above. The train and test accuracies are both dramatically decreased. Probably, it due to the small sample size and the not too flexible model. It's possible, that there are no observation for each factor for each variable in the train dataset.

F)

Let's try a nonlinear model: build a simple tree model and evaluate its performance.

```
set.seed(20220310)
cart <- train(form=form,
  data=caravan_train,
  method = "rpart",
  tuneGrid= expand.grid(cp = 0.01),
  na.action = na.pass,
  trControl = ctrl)


cart$results %>% kable() %>% kable_styling(latex_options = "HOLD_position")
```

cp	Accuracy	Kappa	AccuracySD	KappaSD
0.01	0.9399207	-0.001845	0.0054521	0.0058345

```
# tree train performance
cm_tree <- confusionMatrix(cart$pred$pred,cart$pred$obs)
show_confmat_output(cm_tree, "Simple tree - train")
```

Simple tree - train

	No	Yes
No	876	55
Yes	1	0

	Simple tree - train
Accuracy	0.9399142
Kappa	-0.0021121
AccuracyLower	0.9226796
AccuracyUpper	0.9542962
AccuracyNull	0.9409871
AccuracyPValue	0.5901771
McnemarPValue	0.0000000

	Simple tree - train
Sensitivity	0.9988597
Specificity	0.0000000
Pos Pred Value	0.9409237
Neg Pred Value	0.0000000
Precision	0.9409237
Recall	0.9988597
F1	0.9690265
Prevalence	0.9409871
Detection Rate	0.9399142
Detection Prevalence	0.9989270
Balanced Accuracy	0.4994299

```r
# tree test performance
cm_tree_test <- confusionMatrix(predict(cart,caravan_test),caravan_test$Purchase)
show_confmat_output(cm_tree_test, "Simple tree - test")
```

Simple tree - test

	No	Yes
No	220	12
Yes	0	0

	Simple tree - test
Accuracy	0.9482759
Kappa	0.0000000
AccuracyLower	0.9113920
AccuracyUpper	0.9729911
AccuracyNull	0.9482759
AccuracyPValue	0.5759921
McnemarPValue	0.0014962

	Simple tree - test
Sensitivity	1.0000000
Specificity	0.0000000
Pos Pred Value	0.9482759
Neg Pred Value	NaN
Precision	0.9482759
Recall	1.0000000
F1	0.9734513
Prevalence	0.9482759
Detection Rate	0.9482759
Detection Prevalence	1.0000000
Balanced Accuracy	0.5000000

The tree model reached 0.9399207 accuracy on the training set and 0.9482759 on the test set. These are the best values so far except the benchmark, however, the accuracy is close to the simple benchmark model but lower. We can also notice, that the false negative rate decreased while the false positive rate increased compared to the logit model, and we know that false negative is the worse mistake.

G)

Run a more flexible model (like random forest or GBM). Did it help?

```r
tune_grid <- expand.grid(
  .mtry = 5, # c(5, 6, 7),
  .splitrule = "gini",
  .min.node.size = 15 # c(10, 15)
)
# By default ranger understands that the outcome is binary,
#   thus needs to use 'gini index' to decide split rule
```

```
# getModelInfo("ranger")
set.seed(20220310)
rf_model_p <- train(
   form = formula(paste0("Purchase ~", paste0(vars2, collapse = " + "))),
  method = "ranger",
  data = caravan_train,
  tuneGrid = tune_grid,
  metric="Accuracy",
  trControl = ctrl
)
# Results
rf_model_p$results %>% kable() %>% kable_styling(latex_options = "HOLD_position")# Accuracy 0.94
```

mtry	splitrule	min.node.size	Accuracy	Kappa	AccuracySD	KappaSD
5	gini	15	0.9410077	0	0.0053972	0

```
#Random forest train evaluation
show_confmat_output(confusionMatrix(rf_model_p$pred$pred,rf_model_p$pred$obs),"Random forest - train")
```

Random forest - train

	No	Yes
No	877	55
Yes	0	0

	Random forest - train
Accuracy	0.9409871
Kappa	0.0000000
AccuracyLower	0.9238761
AccuracyUpper	0.9552376
AccuracyNull	0.9409871
AccuracyPValue	0.5357956
McnemarPValue	0.0000000

	Random forest - train
Sensitivity	1.0000000
Specificity	0.0000000
Pos Pred Value	0.9409871
Neg Pred Value	NaN
Precision	0.9409871
Recall	1.0000000
F1	0.9695965
Prevalence	0.9409871
Detection Rate	0.9409871
Detection Prevalence	1.0000000
Balanced Accuracy	0.5000000

```
# Random forest Test evaluation
show_confmat_output(confusionMatrix(predict(rf_model_p,caravan_test),caravan_test$Purchase),"Random fore
```

Random forest - test

	No	Yes
No	220	12
Yes	0	0

	Random forest - test
Accuracy	0.9482759
Kappa	0.0000000
AccuracyLower	0.9113920
AccuracyUpper	0.9729911
AccuracyNull	0.9482759
AccuracyPValue	0.5759921
McnemarPValue	0.0014962

	Random forest - test
Sensitivity	1.0000000
Specificity	0.0000000
Pos Pred Value	0.9482759
Neg Pred Value	NaN
Precision	0.9482759
Recall	1.0000000
F1	0.9734513
Prevalence	0.9482759
Detection Rate	0.9482759
Detection Prevalence	1.0000000
Balanced Accuracy	0.5000000

I run a random forest as a more flexible model, and it improved the performance compared to the tree. the random forest classified everthing as 'No' so the accuracy is the same as the benchmark model (classifying everything as 'No'). This accuracy value of the train and test set are the highest so far compared to the other models (logit, tree), and the false negative prediction rate here is 0 in each cases.

H)

Rerun two of your previous models (a flexible and a less flexible one) on the full train set. Ensure that your test result remains comparable by keeping that dataset intact. (Hint: use the anti_join() function as we did in class.) Interpret your results.

```
caravan_full_train <- anti_join(caravan_data,caravan_test)

set.seed(20220310)
cart_full <- train(form=form,
  data=caravan_full_train,
  method = "rpart",
```

```
  tuneGrid= expand.grid(cp = 0.01),
  na.action = na.pass,
  trControl = ctrl)


cart_full$results %>% kable() %>% kable_styling(latex_options = "HOLD_position")
```

cp	Accuracy	Kappa	AccuracySD	KappaSD
0.01	0.9393071	0	0.0008765	0

```
# tree train performance
cm_tree_full <- confusionMatrix(cart_full$pred$pred,cart_full$pred$obs)
show_confmat_output(cm_tree_full,"Full sample tree - train")
```

Full sample tree - train

	No	Yes
No	5200	336
Yes	0	0

	Full sample tree - train
Accuracy	0.9393064
Kappa	0.0000000
AccuracyLower	0.9326914
AccuracyUpper	0.9454534
AccuracyNull	0.9393064
AccuracyPValue	0.5145110
McnemarPValue	0.0000000

	Full sample tree - train
Sensitivity	1.0000000
Specificity	0.0000000
Pos Pred Value	0.9393064
Neg Pred Value	NaN
Precision	0.9393064
Recall	1.0000000
F1	0.9687034
Prevalence	0.9393064
Detection Rate	0.9393064
Detection Prevalence	1.0000000
Balanced Accuracy	0.5000000

```
# tree test performance
cm_tree_test_full <- confusionMatrix(predict(cart_full,caravan_test),caravan_test$Purchase)
show_confmat_output(cm_tree_test_full,"Full sample tree - test")
```

Full sample tree - test

	No	Yes
No	220	12
Yes	0	0

	Full sample tree - test
Accuracy	0.9482759
Kappa	0.0000000
AccuracyLower	0.9113920
AccuracyUpper	0.9729911
AccuracyNull	0.9482759
AccuracyPValue	0.5759921
McnemarPValue	0.0014962

	Full sample tree - test
Sensitivity	1.0000000
Specificity	0.0000000
Pos Pred Value	0.9482759
Neg Pred Value	NaN
Precision	0.9482759
Recall	1.0000000
F1	0.9734513
Prevalence	0.9482759
Detection Rate	0.9482759
Detection Prevalence	1.0000000
Balanced Accuracy	0.5000000

```
set.seed(20220310)
rf_model_full <- train(
   form = formula(paste0("Purchase ~", paste0(vars2, collapse = " + "))),
  method = "ranger",
  data = caravan_full_train,
  tuneGrid = tune_grid,
  metric="Accuracy",
  trControl = ctrl
)
# Results
rf_model_full$results %>% kable() %>% kable_styling(latex_options = "HOLD_position") # Accuracy
```

mtry	splitrule	min.node.size	Accuracy	Kappa	AccuracySD	KappaSD
5	gini	15	0.9393071	0	0.0008765	0

```
#Random forest train evaluation
show_confmat_output(confusionMatrix(rf_model_full$pred$pred,rf_model_full$pred$obs),"Full sample random
```

Full sample random forest -train

	No	Yes
No	5200	336
Yes	0	0

	Full sample random forest -train
Accuracy	0.9393064
Kappa	0.0000000
AccuracyLower	0.9326914
AccuracyUpper	0.9454534
AccuracyNull	0.9393064
AccuracyPValue	0.5145110
McnemarPValue	0.0000000

	Full sample random forest -train
Sensitivity	1.0000000
Specificity	0.0000000
Pos Pred Value	0.9393064
Neg Pred Value	NaN
Precision	0.9393064
Recall	1.0000000
F1	0.9687034
Prevalence	0.9393064
Detection Rate	0.9393064
Detection Prevalence	1.0000000
Balanced Accuracy	0.5000000

```
# Random forest Test evaluation
show_confmat_output(confusionMatrix(predict(rf_model_full,caravan_test),caravan_test$Purchase),"Full sa
```

Full sample random forest - test

	No	Yes
No	220	12
Yes	0	0

	Full sample random forest - test
Accuracy	0.9482759
Kappa	0.0000000
AccuracyLower	0.9113920
AccuracyUpper	0.9729911
AccuracyNull	0.9482759
AccuracyPValue	0.5759921
McnemarPValue	0.0014962

	Full sample random forest - test
Sensitivity	1.0000000
Specificity	0.0000000
Pos Pred Value	0.9482759
Neg Pred Value	NaN
Precision	0.9482759
Recall	1.0000000
F1	0.9734513
Prevalence	0.9482759
Detection Rate	0.9482759
Detection Prevalence	1.0000000
Balanced Accuracy	0.5000000

We can see on the tables above, that the models trained on the full set did not classified any observations as negative (Yes). So the models performance did not improve compared to the benchmark model (classifying everything as no) because of the extremely low number of 'Yes' cases. Generally, random forest and decision trees are great algorithms, but this example shows than when the frequency of one class is really rare it's really complicated to make good classification models.

Problem 2

A)

Think about an appropriate loss function you can use to evaluate your predictive models. What is the risk (from the business perspective) you would have to take by a wrong prediction?

The root mean squared error (RMSE) would be an appropriate loss function to evaluate the models. It handles error in both direction equally. I think it's appropriate, becasue both over and underestimation is equally bad. If the model underestimates the price, the house will be sold quickly but on cheaper price. If the model overestimates the price, a house needs long time to be sold or it won't be sold at on on that price at all. So it's like a trade off between time and money. Using RMSE, we can tell the applicants that below the estimation the house will be sold quicker, above the estimation the house will be sold on higher price but needs longer time.

B)

Put aside 20% of your data for evaluation purposes (using your chosen loss function). Build a simple benchmark model and evaluate its performance on this hold-out set.

```
real_estate <- read_csv('https://raw.githubusercontent.com/divenyijanos/ceu-ml/main/data/real_estate/rea

set.seed(20220310)

n_obs <- nrow(real_estate)
test_share <- 0.2

test_indices <- sample(seq(n_obs), floor(test_share * n_obs))
real_estate_test <- slice(real_estate, test_indices)
real_estate_train <- slice(real_estate, -test_indices)
```

I build a benchmark prediction model, which is a simple linear regression with the age of the house as explanatory variable.

```
benchmark <- lm(house_price_of_unit_area ~ house_age, data = real_estate_train)
```

```
# RMSE train
RMSE(benchmark$fitted.values,real_estate_train$house_price_of_unit_area)
```

```
## [1] 13.53866
```

```
# RMSE test
RMSE(predict(benchmark,real_estate_test),real_estate_test$house_price_of_unit_area)
```

```
## [1] 12.2489
```

The RMSE on the train and test sets are 13.5386578 and 12.2489011.

C)

Build a simple linear regression model and evaluate its performance. Would you launch your evaluator web app using this model?

```
vars <- names(real_estate)[2:7]
formula <-paste0("house_price_of_unit_area~",paste( as.list(vars),collapse = '+',sep=''))

reg1 <- lm(formula = formula, data=real_estate_train)
# Evaluate train
RMSE(reg1$fitted.values,real_estate_train$house_price_of_unit_area)
```

```
## [1] 8.843894
```

```
# Evaluate test
RMSE(predict(reg1,real_estate_test),real_estate_test$house_price_of_unit_area)
```

```
## [1] 8.546549
```

This model is much better than the benchmark (~ 30% lower RMSE) but I would still not use this as the model for my app, because we can try to get better model. I see room for feature engineering, other functional forms and for more flexible models as well, which can improve the performance.

D)

Try to improve your model. Take multiple approaches (e.g. feature engineering, more flexible models, stacking, etc.) and document their successes.

The main feature of the regression is that it's linear, while in reality relationship between variables is often not linear. Also, we do not use the transaction date, latitude and longitude variables properly, so these can certainly improved.

Firstly, the latitude and longitude values are not so meaningful about the location within a city. To get a more meaningful measure, I searched the geo code of New Tapipei City's central district: Banqiao District. I subsreacted the latitude and longitude parameter from the central district parameter and took the absolute value of them. These *lat_from_centre* and *lon_from_centre* show the distance from the central district, so it's easy to understand and inpret, moreover I expect improvement in the model by them.

Secondly, I converted the transaction date to correct date format. It can also improve the model by detecting any pattern. I also added the month as factor, to detect any seasonality. To get more intuition about the data, I visulaized the relationship between price and other variables.

```
# Lat and lon values

lat<-25.011262
lon<-121.445867
```

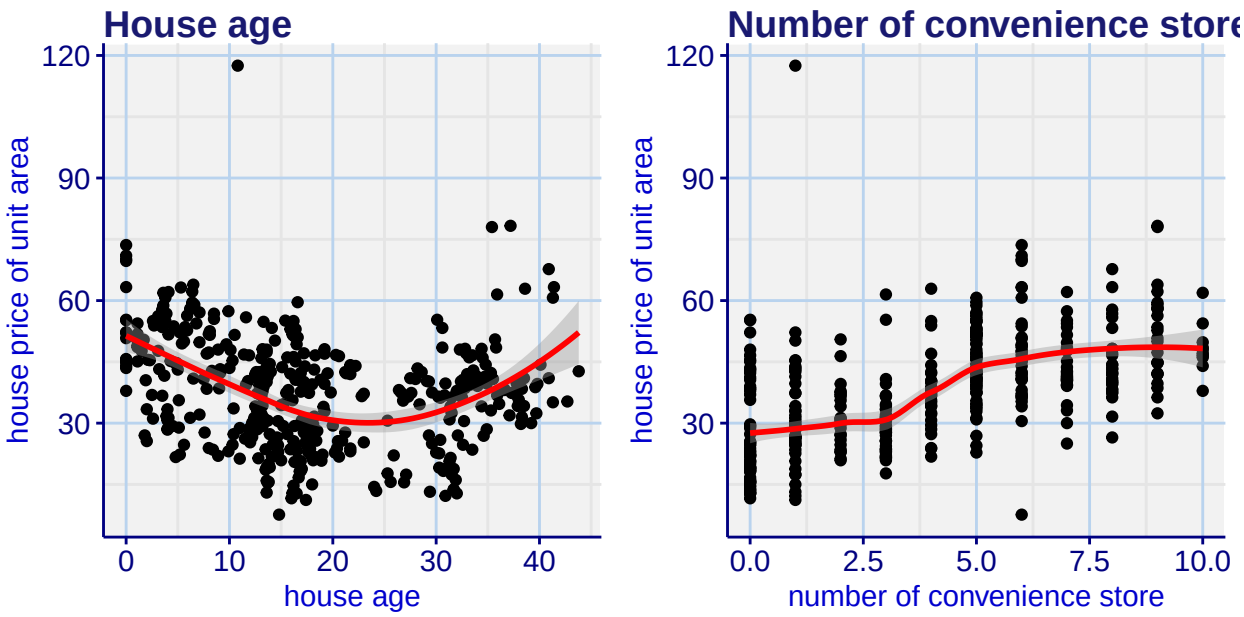
```
real_estate <- real_estate %>%  mutate(
  lat_from_centre = abs(latitude - lat),
  lon_from_centre = abs(longitude - lon),
  date_correct =ymd(paste(floor(transaction_date), as.character(round(transaction_date %% 1 * 12+1 ),0)
  date_month_factor = as.factor(month(ymd(paste(floor(transaction_date), as.character(round(transaction_
  house_age_sq = house_age^2,
  distance_to_the_nearest_MRT_station_sq=distance_to_the_nearest_MRT_station^2
)


real_estate_test <- slice(real_estate, test_indices)
real_estate_train <- slice(real_estate, -test_indices)

p1<-ggplot(real_estate,aes(x=house_age, y=house_price_of_unit_area))+
  geom_point()+
  geom_smooth(methos="loess", color="red")+
  labs(title = "House age",y="house price of unit area",x="house age")

p2<-ggplot(real_estate,aes(x=number_of_convenience_stores, y=house_price_of_unit_area))+
  geom_point()+
  geom_smooth(methos="loess", color="red")+
  labs(title = "Number of convenience stores",y="house price of unit area",x="number of convenience sto

ggarrange(p1,p2,nrow=1)
```

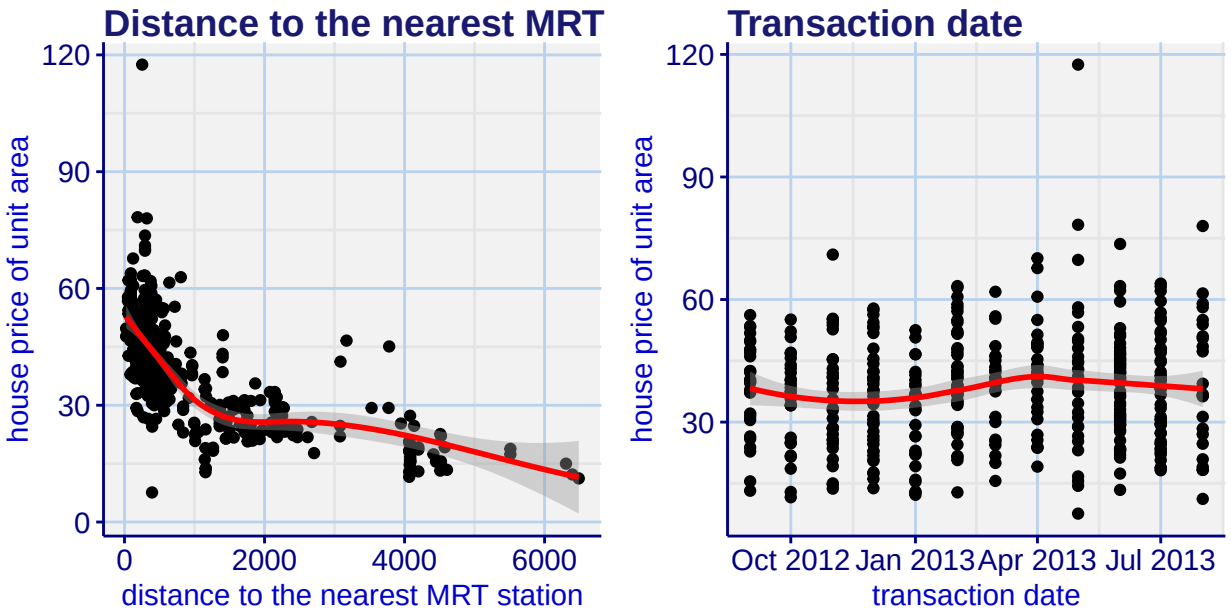



```
p3<-ggplot(real_estate,aes(x=distance_to_the_nearest_MRT_station, y=house_price_of_unit_area))+
  geom_point()+
  geom_smooth(methos="loess", color="red")+
  labs(title = "Distance to the nearest MRT",y="house price of unit area",x="distance to the nearest MR


p4<-ggplot(real_estate,aes(x=date_correct, y=house_price_of_unit_area))+
  geom_point()+
```

```
  geom_smooth(methos="loess", color="red")+
  labs(title = "Transaction date",y="house price of unit area",x="transaction date")

ggarrange(p3,p4,nrow=1)
```



To better capture the functional for I run a regression with squared variables and adding the new feature engineered geocodes and transaction dates.

```
real_estate_test <- slice(real_estate, test_indices)
real_estate_train <- slice(real_estate, -test_indices)

reg2 <-lm(house_price_of_unit_area ~ house_age+ I(house_age^2) + distance_to_the_nearest_MRT_station+nu


# Evaluate train
RMSE(reg2$fitted.values,real_estate_train$house_price_of_unit_area)
```

```
## [1] 7.988436
```

```
# Evaluate test
RMSE(predict(reg2,real_estate_test),real_estate_test$house_price_of_unit_area)
```

```
## [1] 7.297354
```

The train and test RMSE of the improved regression are 7.9884357 and 7.2973537 . There is 0.8554581 and 1.249195 improvement on the train and test sets respectively. However, we can try more flexible models as well.

Random forest is a great option because we do not have to care about the functional form, so I also trained a random forest using the feature engineered data. By training the random forest, I have tried several but it did not change much. Due to the small dataset, the seed / randomization change a lot on the vales.

```
# Random forest
tune_grid <- expand.grid(
  .mtry = 5, # c(3,4,5, 6, 7),
 .splitrule = "variance",
```

```
  .min.node.size = 15 # c(10, 15)
)
#

set.seed(20220310)
formula_corrected <-"house_price_of_unit_area~date_correct+house_age+distance_to_the_nearest_MRT_statio

rf_realestate <-train(
   form = formula(formula_corrected),
  method = "ranger",
  data = real_estate_train,
  tuneGrid = tune_grid,
  metric="RMSE",
  trControl = ctrl
)


rf_realestate$results$RMSE
```

```
## [1] 7.325345
```

```
RMSE(predict(rf_realestate,real_estate_train),real_estate_train$house_price_of_unit_area)
```

```
## [1] 5.081511
```

Surprisingly, the test RMSE of the random forest is much lower than the train RMSE. It is probably not because of overfit, because overfitting would result higher test RMSE than train. It is probably by accident, and because of the small sample. The real estate test set has only 82 observations, so this test RMSE value is not so well-founded. The train RMSE of the random forest improved 0.66 compared to the previous regression while the test improved 2.22.

I also trained a GBM similarly to the random forest, on the feature engineered data and trying different hyperparameters.

```
gbm_grid <-  expand.grid(interaction.depth = 5, # complexity of the tree
                         n.trees = 100, # number of iterations, i.e. trees
                         shrinkage = 0.05, # learning rate: how quickly the algorithm adapts
                         n.minobsinnode = 10 # the minimum number of training set samples in a node to
)

set.seed(20220310)
gbm_model <- train(formula(formula_corrected),
                   data = real_estate_train,
                   method = "gbm",
                   trControl = ctrl,
                   verbose = FALSE,
                   tuneGrid = gbm_grid)

# GBM train RMSE
gbm_model$results$RMSE
```

```
## [1] 7.461572
```

```
# GBM test RMSE
RMSE(predict(gbm_model,real_estate_test),real_estate_test$house_price_of_unit_area)
```

```
## [1] 6.386305
```

The test RMSE of the GBM is nearly the same as the random forest's, while the GBM's test RMSE is higher than the random forest's but is still lower than the train, so it likely no overfitting. Both the GBM and random foest has much lower RMSE on each set than the regressions.

E)

Would you launch your web app now? What options you might have to further improve the prediction performance?

The more flexible models (random forest and GBM) perform better than the regressions and I find them more reliable as well. However, on such a small dataset like this, it is hard to make a good model. The RMSE improved a lot compared to the benchmark model, but it's still 4-7 times 10000 New Taiwan Dollar/Ping (where Ping is a local unit, 1 Ping = 3.3 meter squared).

To improve the models, I would definitely collect more data to train my models before launching the web app. The data collection would be in both dimensions aka more variables like rooms, exact location, district, condition ... and more observations as well. Webscraping could be an easy solution for this.