ML-Tools-Assignment-1

Gyongyver Kamenar (2103380)

3/14/2022

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

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.

How accurately we can predict, that the caravan will purchase insurance. 100% accurately 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()
```

x
1.059013

```
caravan_train %>% group_by(Purchase) %>% count() %>% kable()
```

Purchase	n
No	877
Yes	55

The average as 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.

```
set.seed(132456)

variables <- colnames(caravan_data[,1:85])

form <- formula(paste0("Purchase ~", paste0(variables, 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()# Accuracy
```

parameter	Accuracy	Kappa	AccuracySD	KappaSD
none	0.9196726	-0.0071954	0.0232684	0.0725444

```
#logit_model$pred
```

I trained a logit model with all variables, and it has 0.9196726 accuracy, which is quite bad compared to the benchmark model. So let's see the further details of the model and the predictions.

```
# Train evaluation
cm_logit<-confusionMatrix(logit_model$pred$pred,logit_model$pred$obs)
cm_logit
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  856  54
##        Yes  21   1
##
##                Accuracy : 0.9195
##                  95% CI : (0.9002, 0.9362)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.9967820
##
##                   Kappa : -0.008
##
##  Mcnemar's Test P-Value : 0.0002199
##
##             Sensitivity : 0.97605
```

```
##             Specificity : 0.01818
##          Pos Pred Value : 0.94066
##          Neg Pred Value : 0.04545
##              Prevalence : 0.94099
##          Detection Rate : 0.91845
##    Detection Prevalence : 0.97639
##       Balanced Accuracy : 0.49712
##
##        'Positive' Class : No
##
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  216  12
##        Yes   4   0
##
##                Accuracy : 0.931
##                  95% CI : (0.8904, 0.9601)
##     No Information Rate : 0.9483
##     P-Value [Acc > NIR] : 0.90447
##
##                   Kappa : -0.0265
##
##  Mcnemar's Test P-Value : 0.08012
##
##             Sensitivity : 0.9818
##             Specificity : 0.0000
##          Pos Pred Value : 0.9474
##          Neg Pred Value : 0.0000
##              Prevalence : 0.9483
##          Detection Rate : 0.9310
##    Detection Prevalence : 0.9828
##       Balanced Accuracy : 0.4909
##
##        'Positive' Class : No
##
```

On the train dataset, the logit model classified 857 items correctly out of the 932. The logit predicted 21 'Yes' while actually these do not have insurence (FN) and predicted 54 '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 0.9760547 while the specificity is 0.0181818. This means, that the model classified 1.8181818% of the negative cases correctly. The positive predictive value is 0.9406593 which is not so bad, however, the negative predictive value is 0.0454545 which is really low. It basically means, that the model's negative classifications is in only 4.5454545% true.

On the test dataset, the accuracy is 0.9310345. The FN rate is 0.0172414 while the FP rate is `rcm_logit_test$table[3]/nrow(caravan_test)`. 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?

```
#summary(caravan_data)

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

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(variables, collapse = " + "))),
    method    = "glm",
    data      = caravan_train,
    family    = binomial,
     trControl = ctrl
  )

logit_model_factor$results # Accurary 0.78
```

```
##   parameter  Accuracy       Kappa AccuracySD   KappaSD
## 1      none 0.7780977 -0.01388793 0.08220332 0.0947437
```

```
# Train evaluation
confusionMatrix(logit_model_factor$pred$pred,logit_model_factor$pred$obs)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  717  47
##        Yes 160   8
##
##                Accuracy : 0.7779
##                  95% CI : (0.7498, 0.8042)
```

```
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0188
##
##  Mcnemar's Test P-Value : 6.997e-15
##
##             Sensitivity : 0.81756
##             Specificity : 0.14545
##          Pos Pred Value : 0.93848
##          Neg Pred Value : 0.04762
##              Prevalence : 0.94099
##          Detection Rate : 0.76931
##    Detection Prevalence : 0.81974
##       Balanced Accuracy : 0.48151
##
##        'Positive' Class : No
##
```

```
# Test evaluation

confusionMatrix(predict(logit_model_factor,caravan_test),caravan_test$Purchase)# Test Accuracy 0.85
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  194   8
##        Yes  26   4
##
##                Accuracy : 0.8534
##                  95% CI : (0.8013, 0.8963)
##     No Information Rate : 0.9483
##     P-Value [Acc > NIR] : 1.000000
##
##                   Kappa : 0.1259
##
##  Mcnemar's Test P-Value : 0.003551
##
##             Sensitivity : 0.8818
##             Specificity : 0.3333
##          Pos Pred Value : 0.9604
##          Neg Pred Value : 0.1333
##              Prevalence : 0.9483
##          Detection Rate : 0.8362
##    Detection Prevalence : 0.8707
##       Balanced Accuracy : 0.6076
##
##        'Positive' Class : No
##
```

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()
```

cp	Accuracy	Kappa	AccuracySD	KappaSD
0.01	0.9399207	-0.001845	0.0054521	0.0058345

```
cart$results$Accuracy
```

```
## [1] 0.9399207
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  876  55
##        Yes   1   0
##
##                Accuracy : 0.9399
##                  95% CI : (0.9227, 0.9543)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.5902
##
##                   Kappa : -0.0021
##
##  Mcnemar's Test P-Value : 1.417e-12
##
##             Sensitivity : 0.9989
##             Specificity : 0.0000
##          Pos Pred Value : 0.9409
##          Neg Pred Value : 0.0000
##              Prevalence : 0.9410
##          Detection Rate : 0.9399
##    Detection Prevalence : 0.9989
##       Balanced Accuracy : 0.4994
##
##        'Positive' Class : No
##
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  220  12
##        Yes   0   0
##
##                Accuracy : 0.9483
##                  95% CI : (0.9114, 0.973)
##     No Information Rate : 0.9483
##     P-Value [Acc > NIR] : 0.575992
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 0.001496
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9483
##          Neg Pred Value :    NaN
##              Prevalence : 0.9483
##          Detection Rate : 0.9483
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

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, however, the accuracy is close to the simple benchmark model. 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?

```
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(variables, collapse = " + "))),
  method = "ranger",
  data = caravan_train,
  tuneGrid = tune_grid,
  metric="Accuracy",
  trControl = ctrl
)
# Results
```

```
rf_model_p$results %>% kable() # 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
confusionMatrix(rf_model_p$pred$pred,rf_model_p$pred$obs)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  877  55
##        Yes   0   0
##
##                Accuracy : 0.941
##                  95% CI : (0.9239, 0.9552)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.5358
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 3.305e-13
##
##             Sensitivity : 1.000
##             Specificity : 0.000
##          Pos Pred Value : 0.941
##          Neg Pred Value :   NaN
##              Prevalence : 0.941
##          Detection Rate : 0.941
##    Detection Prevalence : 1.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : No
##
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  220  12
##        Yes   0   0
##
##                Accuracy : 0.9483
##                  95% CI : (0.9114, 0.973)
##     No Information Rate : 0.9483
##     P-Value [Acc > NIR] : 0.575992
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 0.001496
##
```

```
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9483
##          Neg Pred Value :    NaN
##              Prevalence : 0.9483
##          Detection Rate : 0.9483
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

I run a random ofrest as a more flexible model, and it improved the performance. The accuracy value of the train and test set are the highest so far, and the false negative prediction rate 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()
```

cp	Accuracy	Kappa	AccuracySD	KappaSD
0.01	0.9393071	0	0.0008765	0

```
cart_full$results$Accuracy
```

```
## [1] 0.9393071
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  5200  336
##        Yes    0    0
##
##                Accuracy : 0.9393
##                  95% CI : (0.9327, 0.9455)
##     No Information Rate : 0.9393
##     P-Value [Acc > NIR] : 0.5145
##
```

```
##                  Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9393
##          Neg Pred Value :    NaN
##              Prevalence : 0.9393
##          Detection Rate : 0.9393
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  220  12
##        Yes   0   0
##
##                Accuracy : 0.9483
##                  95% CI : (0.9114, 0.973)
##     No Information Rate : 0.9483
##     P-Value [Acc > NIR] : 0.575992
##
##                  Kappa : 0
##
##  Mcnemar's Test P-Value : 0.001496
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9483
##          Neg Pred Value :    NaN
##              Prevalence : 0.9483
##          Detection Rate : 0.9483
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

```
set.seed(20220310)
rf_model_full <- train(
   form = formula(paste0("Purchase ~", paste0(variables, collapse = " + "))),
  method = "ranger",
  data = caravan_full_train,
  tuneGrid = tune_grid,
  metric="Accuracy",
```

```
  trControl = ctrl
)
# Results
rf_model_full$results %>% kable() # Accuracy
```

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

```
#Random forest train evaluation
confusionMatrix(rf_model_full$pred$pred,rf_model_full$pred$obs)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  5200  336
##        Yes    0    0
##
##                Accuracy : 0.9393
##                  95% CI : (0.9327, 0.9455)
##     No Information Rate : 0.9393
##     P-Value [Acc > NIR] : 0.5145
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9393
##          Neg Pred Value :    NaN
##              Prevalence : 0.9393
##          Detection Rate : 0.9393
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

```
# Random forest Test evaluation
confusionMatrix(predict(rf_model_full,caravan_test),caravan_test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  220  12
##        Yes   0   0
##
##                Accuracy : 0.9483
##                  95% CI : (0.9114, 0.973)
##     No Information Rate : 0.9483
##     P-Value [Acc > NIR] : 0.575992
##
##                   Kappa : 0
##
```

```
##  Mcnemar's Test P-Value : 0.001496
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9483
##          Neg Pred Value :    NaN
##              Prevalence : 0.9483
##          Detection Rate : 0.9483
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

The models trained on the full set did not classified any observations as negative (Yes).

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 ()

D)

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

E)

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