Homework 7

2025-10-09

Boilerplate

We load the necessary libraries:

```
library(printr)     # pretty print for Rmd
library(lubridate)  # dates
library(ggplot2)    # plots
library(dplyr)      # dataframes
library(tidyr)
library(tidyverse)
library(recipes)
library(caret)
library(tree)
library(randomForest)
library(stats)
library(MASS)      # step models
library(Metrics)

# set seed for reproducibility
set.seed(42)
```

Question 10.1

Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using

a. a regression tree model, and
b. a random forest model.

In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

First let's load the data:

```
df <- read.table("./data 10.1/uscrime.txt", header =TRUE)
```

Note: As we know, our data set is extremely limited, therefore I will do 10-fold cross validation on the final model, but am not going to be creating a final test set.

Comparison to Homework 5

From Homework 5, my best model on all the training data had an adjusted R^2 of 0.8514 and in 10-fold cross validation had R^2 of 0.7953447, so those are the baselines I will be evaluating against. I am going to add the same features as I used previously.

```
crime_stats <- df %>% mutate(
  InvProb = 1 / Prob,
  LF2 = LF^2,
  NW2 = NW^2
)
```

I'm going to compare our standard point just out of curiosity.

```
crime_test <- data.frame(
  M    = 14.0, So   = 0    , Ed   = 10.0, Po1    = 12.0,
  Po2  = 15.5, LF   = 0.640, M.F  = 94.0, Pop    = 150,
  NW   = 1.1 , U1   = 0.120, U2   = 3.6 , Wealth = 3200,
  Ineq = 20.1, Prob = 0.04 , Time = 39.0
) %>% mutate(
  InvProb = 1 / Prob,
  LF2 = LF^2,
  NW2 = NW^2
)
```

10.1.A - Tree Regression

Let's start by just training both models and see what we get.

Note: I got the CV from tree's man page[1]; however, they neglected to provide any description of the output. So I asked ChatGPT to explain the output. Its stated that the size is the size of the pruned tree so we need to choose the size that minimizes the deviance. To do this, I'm going to train 100 trees and choose the best size of the best tree.

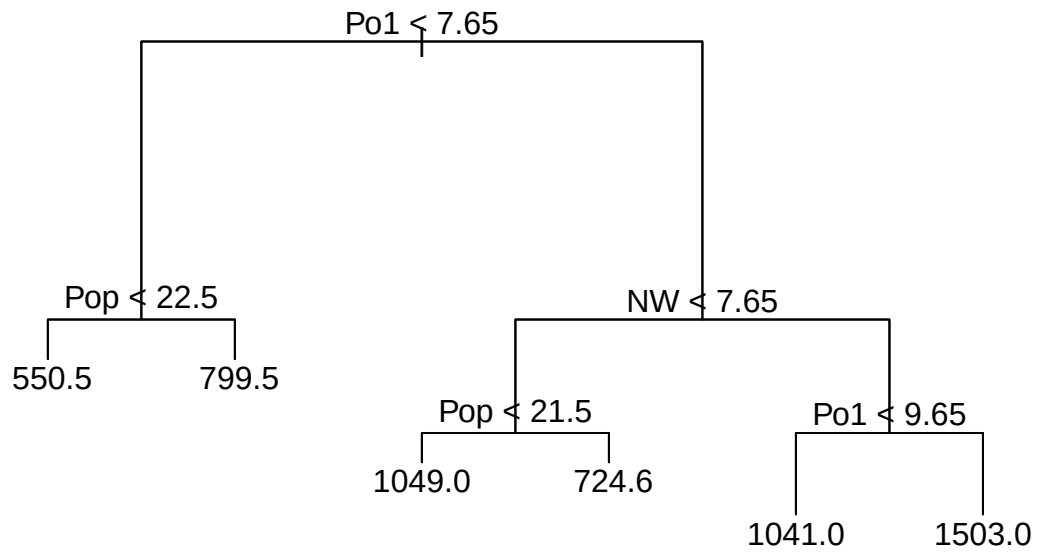
```
tree_model <- list()
tree_cv <- list()

for (i in seq(100))
{
  # Train the tree
  tree_out <- tree(Crime ~ ., crime_stats)

  # assign to aggregation variables
  tree_cv[[i]] <- cv.tree(tree_out)
  tree_model[[i]] <- tree_out
}

# extract the deviation from each run
dev_mat <- sapply(tree_cv, function(x) { x$dev } )

# labeling for distinguishability
colnames(dev_mat) <- paste0("tree_", seq_along(tree_cv))
rownames(dev_mat) <- tree_cv[[1]]$size

# Let's get the min index of the smallest value
# https://stackoverflow.com/a/17551974/1543042
smallest_index <- which(dev_mat == min(dev_mat), arr.ind = TRUE)

row_name <- rownames(dev_mat)[smallest_index[,"row"]]
col_name <- colnames(dev_mat)[smallest_index[,"col"]]

data.frame(
  depth = row_name,
  tree = col_name,
  value = dev_mat[smallest_index]
)
```

depth	tree	value
6	tree_26	5366854

[1] https://cran.r-project.org/web/packages/tree/refman/tree.html#cv.tree

We have our tree, let's see what it looks like

```
pruned_tree <- prune.tree(tree_model[[26]], best = 6)
plot(pruned_tree)
text(pruned_tree)
```



So that's a simple model... it does have features we know are important such as police, population and percent of the population that is non-white. Let's look at some metrics to compare to Homeworks 5 & 6.

```
tree_preds <- predict(pruned_tree, crime_stats)
cor(tree_preds, crime_stats$Crime)^2
```

```
## [1] 0.7074149
```

```
mae(crime_stats$Crime, tree_preds)
```

```
## [1] 153.6278
```

```
rmse(crime_stats$Crime, tree_preds)
```

```
## [1] 206.9668
```

That's significantly worse than any of the models we have seen so far. That hardly surprising given that there are only 6 possible values we can predict. Let's see what we predict for our test point

```
predict(pruned_tree, crime_test)
```

```
##     1
## 724.6
```

That is significantly higher than I predicted in Homework 5 (490) or Homework 6 (614).

10.1.B - Random Forest Regression

Now we turn our attention to Random Forest regression, which answers the question what if we took a lot of really bad models and see what we get? So let's do that

```
forest <- randomForest(
  formula=Crime ~ .,
  data=crime_stats
)

forest_preds <- predict(forest, crime_stats)
cor(forest_preds, crime_stats$Crime)^2
```

```
## [1] 0.9444974
```

```
mae(crime_stats$Crime, forest_preds)
```

```
## [1] 82.58764
```

```
rmse(crime_stats$Crime, forest_preds)
```

```
## [1] 121.8766
```

The forest definitely does a better job at predicting the crime rate; however, it is still orders of magnitude worse MAE and RMSE. I ran this several times and these are representative, the MAE was always between 80 and 85.

Just for completeness let's see what the prediction for our test city is

```
predict(forest, crime_test)
```

```
##        1
## 1186.416
```

That is the highest prediction we have seen from any of the models. It is definitely interesting that this is significantly higher than our tree while the overall error is lower.

Question 10.2

> Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.

An example of where I have used logistic regression in my work is in the realm of unacceptable off-power mode drain from 12V batteries. We want to encode whether an observed drain is unacceptably high; however, the threshold that we are willing to accept depends on a few factors:

1. The vehicle propulsion system - EVs both have a more off-power feature sets [thermal runaway detection, and customer facing features] and a ready source of energy from the high voltage battery compared to internal combustion vehicles. Therefore, our acceptable threshold is higher
2. System architecture - My company has multiple system architectures in the field currently, and unfortunately some on average perform more poorly than others and so we take that into account when determining if the drain is unacceptable
3. Off-time - As the vehicle is off for a longer period of time features are disabled to save energy, therefore a drain rate that is acceptable if the vehicle was off for 12 hours, is a problem if it instead has been off for 2 weeks.

Unfortunately due to the inherent non-linearities in the system, we eventually decided that the prediction accuracy using logistic regression was below acceptable levels and decided to use a neural network. This demonstrates the explainability-accuracy conflict that Prof. Sokol discussed in the lecture.

Question 10.3

a. Using the GermanCredit data set germancredit.txt from https://archive.ics.uci.edu/static/public/144/statlog+german+credit+data.zip/ (description at https://archive.ics.uci.edu/dataset/144/statlog+german+credit+data), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use family=binomial(link="logit") in your glm function call.

b. Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.

First let's load the data. I also retrieved the columns meanings because I wasn't a fan of just V_n.

```
german_credit <- read.table("./data 10.3/germancredit.txt", header = FALSE)
colnames(german_credit) <- c(
  "Checking_Account_Status", "Duration",
  "Credit_history", "Purpose",
  "Credit_amount", "Savings",
  "Employment", "Normalized_Payment_Size",
  "Personal_status_and_sex", "Other_debtors_Guarantors",
  "Time_In_Current_Residence", "Property",
  "Age", "Other_installment_plans",
  "Housing", "Number_of_existing_credits_at_this_bank",
  "Job", "Number_of_people_liable",
  "Telephone", "foreign_worker",
  "Credit_Worthiness"
)
```

Interestingly, I thought we had to one-hot encode our categorical variables, but that's handled automatically by `glm` which is a nice quality of life feature. We do need to switch the credit worthiness encoding to 0 and 1.

```
# as.factor because caret::train complained
german_credit$Credit_Worthiness = as.factor(2 - german_credit$Credit_Worthiness)
```

10.3.A - Logistic Regression Model

For this dataset we definitely have enough to train-test split. Useful package[2]

```
data_split <- rsample::initial_split(german_credit, prop = 0.8)
training_data <- rsample::training(data_split)
test_data <- rsample::testing(data_split)
```

Let's create a model with all the factors and see what comes out

```
train_control <- trainControl(method = "cv", number = 10)
full_feature_model <- train(
  Credit_Worthiness ~ .,
  data = training_data,
  method = "glm",
  family = binomial(link="logit"),
  trControl = train_control
)

coeffs <- tibble::rownames_to_column(
  as.data.frame(full_feature_model$finalModel$coefficients),
  "x"
)
colnames(coeffs) <- c("coefficient_names", "coefficient_vals")


confusionMatrix(full_feature_model)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    0    1
##          0 15.4  9.8
##          1 14.4 60.5
##
##  Accuracy (average) : 0.7588
```

[2]https://rsample.tidymodels.org/reference/initial_split.html

And let's print out the coefficients for the full model

```
knitr::kable(coeffs)
```

coefficient_names	coefficient_vals
(Intercept)	0.2747922
Checking_Account_StatusA12	0.1932078
Checking_Account_StatusA13	0.7521855
Checking_Account_StatusA14	1.7375346
Duration	-0.0389174
Credit_historyA31	0.3001940
Credit_historyA32	0.5932216
Credit_historyA33	1.1594366
Credit_historyA34	1.8154466
PurposeA41	1.9330935
PurposeA410	1.3960133
PurposeA42	0.7344541
PurposeA43	0.9862148
PurposeA44	0.4401155
PurposeA45	0.2701092
PurposeA46	-0.5790663
PurposeA48	1.3478383
PurposeA49	0.9971303
Credit_amount	-0.0000992
SavingsA62	0.5055936
SavingsA63	0.3697661
SavingsA64	1.4222716
SavingsA65	1.4367399
EmploymentA72	0.1624238
EmploymentA73	0.0283523
EmploymentA74	0.5407204
EmploymentA75	0.5020587
Normalized_Payment_Size	-0.2623975
Personal_status_and_sexA92	0.2167910
Personal_status_and_sexA93	0.8221316
Personal_status_and_sexA94	0.3448071
Other_debtors_GuarantorsA102	-0.3874762
Other_debtors_GuarantorsA103	1.3202257
Time_In_Current_Residence	-0.0543317
PropertyA122	-0.2016314
PropertyA123	-0.1810248
PropertyA124	-1.0394039
Age	0.0089162
Other_installment_plansA142	0.2019577
Other_installment_plansA143	0.7075943
HousingA152	0.2563497
HousingA153	1.1741363
Number_of_existing_credits_at_this_bank	-0.4793369
JobA172	-0.7820858
JobA173	-0.7488405
JobA174	-0.8516361
Number_of_people_liable	-0.2179852
TelephoneA192	0.3811073
foreign_workerA202	1.1533221

That's quite a set! As we saw from the confusion matrix our full model is more likely to return false positive than true negative. That really drives home the point for us needing a good threshold!

10.3.B - Threshold setting

So now we want to find the threshold so that we minimize overall cost given that a false positive is 5 times worse than a false negative. To set our threshold we need to extract out the probabilities of the model rather than the preset. Since this only has two classes the columns are complements and we only need to use one of them.

```
label_and_prob <- data.frame(
  label = training_data$Credit_Worthiness
)

label_and_prob$prob <- predict(
  full_feature_model,
  training_data,
  type = "prob"
)[, "1"]
```

So, we can convert our requirement into the expression. Because this is choosing the hyperparameter we want to exclude the test set.

```
cost <- function(p)
{
  # predicted positive
  M <- label_and_prob$prob >= p

  # actually positive
  L <- label_and_prob$label == 1

  # false positive
  false_pos <- (1-L)*M

  # false negative
  false_neg <- L*(1-M)

  sum(5*false_pos+1*false_neg)
}
```

Now we just need to minimize it over the interval $p \in [0, 1]$. R has the `optimize` function to do that[3].

```
xmin <- optimize(cost, c(0, 1))
pmin <- xmin$minimum

pmin
```

```
## [1] 0.807141
```

[3] https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/optimize

Because false positives are costly, our probability threshold is driven above 50% so that we accept fewer applicants. Let's see what that does to the confusion matrix.

```
cm <- confusionMatrix(
  data=as.factor(label_and_prob$prob >= pmin),
  reference=as.factor(label_and_prob$label == 1)
)
round(100 * cm$table / sum(cm$table), 1)
```

Prediction/Reference	FALSE	TRUE
FALSE	26.4	26.0
TRUE	3.4	44.2

I'm quite surprised that the confusion matrix is that unbalanced! We had to give up ~25% of our true positives to drive down the false positive rate. Let's run this on the test set.

```
label_and_prob_test <- data.frame(
  label = test_data$Credit_Worthiness
)

label_and_prob_test$prob <- predict(
  full_feature_model,
  test_data,
  type = "prob"
)[, "1"]

cm <- confusionMatrix(
  data=as.factor(label_and_prob_test$prob >= pmin),
  reference=as.factor(label_and_prob_test$label == 1)
)
round(100 * cm$table / sum(cm$table), 1)
```

Prediction/Reference	FALSE	TRUE
FALSE	22.5	26.5
TRUE	8.5	42.5

Very similar performance.