Data622_Assignment_3

Natalie Kalukeerthie and Anna Moy

2025-11-01

The area of expertise for this analysis is marketing analytics and customer prediction. The dataset focuses on predicting whether a client subscribes to a bank's term deposit following a marketing campaign—a classic classification problem in customer behavior modeling. Within this field, algorithms like Decision Trees, Random Forests, and SVMs are widely used to identify key influencing factors such as client age, previous contacts, and communication methods.

Our results directly support this domain: the Random Forest model provided the best overall accuracy and balance between recall and precision, making it a strong choice for predicting customer responses in marketing scenarios. The interpretability of tree-based models also helps marketers understand which variables most influence conversion rates—insights essential for targeting strategies and campaign optimization.

Assignment 3

```
library(tidyverse)
library(tidymodels)
library(caret)        # optional
library(rpart)        # for decision trees
library(ranger)       # random forest engine
library(kernlab)      # SVM engine for parsnip (sum_rbf)
library(yardstick)
library(rpart.plot)


#import full bank dataset

# Import the CSV file from github
bank <- read_csv2("https://raw.githubusercontent.com/nk014914/Data622/refs/heads/main/bank-full.csv", sl
```

First, we'll look at the structure of the data.

```
## Rows: 45,211
## Columns: 17
## $ age       <dbl> 58, 44, 33, 47, 33, 35, 28, 42, 58, 43, 41, 29, 53, 58, 57, ~
## $ job       <chr> "management", "technician", "entrepreneur", "blue-collar", "~
## $ marital   <chr> "married", "single", "married", "married", "single", "marrie~
## $ education <chr> "tertiary", "secondary", "secondary", "unknown", "unknown", ~
## $ default   <chr> "no", "no", "no", "no", "no", "no", "no", "yes", "no", "no",~
## $ balance   <dbl> 2143, 29, 2, 1506, 1, 231, 447, 2, 121, 593, 270, 390, 6, 71~
## $ housing   <chr> "yes", "yes", "yes", "yes", "no", "yes", "yes", "yes", "yes"~
```

```
## $ loan      <chr> "no", "no", "yes", "no", "no", "no", "yes", "no", "no", "no"~
## $ contact   <chr> "unknown", "unknown", "unknown", "unknown", "unknown", "unkn~
## $ day       <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ month     <chr> "may", "may", "may", "may", "may", "may", "may", "may", "may~
## $ duration  <dbl> 261, 151, 76, 92, 198, 139, 217, 380, 50, 55, 222, 137, 517,~
## $ campaign  <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ pdays     <dbl> -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, ~
## $ previous  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ poutcome  <chr> "unknown", "unknown", "unknown", "unknown", "unknown", "unkn~
## $ y         <chr> "no", "no", "no", "no", "no", "no", "no", "no", "no", "no", ~
```

```
##       age            job              marital           education
##  Min.   :18.00   Length:45211       Length:45211       Length:45211
##  1st Qu.:33.00   Class :character   Class :character   Class :character
##  Median :39.00   Mode  :character   Mode  :character   Mode  :character
##  Mean   :40.94
##  3rd Qu.:48.00
##  Max.   :95.00
##    default            balance         housing             loan
##  Length:45211       Min.   : -8019   Length:45211       Length:45211
##  Class :character   1st Qu.:    72   Class :character   Class :character
##  Mode  :character   Median :   448   Mode  :character   Mode  :character
##                     Mean   :  1362
##                     3rd Qu.:  1428
##                     Max.   :102127
##    contact               day           month              duration
##  Length:45211       Min.   : 1.00   Length:45211       Min.   :   0.0
##  Class :character   1st Qu.: 8.00   Class :character   1st Qu.: 103.0
##  Mode  :character   Median :16.00   Mode  :character   Median : 180.0
##                     Mean   :15.81                      Mean   : 258.2
##                     3rd Qu.:21.00                      3rd Qu.: 319.0
##                     Max.   :31.00                      Max.   :4918.0
##     campaign          pdays          previous          poutcome
##  Min.   : 1.000   Min.   : -1.0   Min.   :  0.0000   Length:45211
##  1st Qu.: 1.000   1st Qu.: -1.0   1st Qu.:  0.0000   Class :character
##  Median : 2.000   Median : -1.0   Median :  0.0000   Mode  :character
##  Mean   : 2.764   Mean   : 40.2   Mean   :  0.5803
##  3rd Qu.: 3.000   3rd Qu.: -1.0   3rd Qu.:  0.0000
##  Max.   :63.000   Max.   :871.0   Max.   :275.0000
##       y
##  Length:45211
##  Class :character
##  Mode  :character
##
##
##
```

```
## [1] 0
```

Our chosen dataset,'Bank-Full', has 45,211 rows and 17 columns. It contains both numerical variables (e.g., age, balance, duration) and categorical variables (e.g., job, marital status, education). Our target variable is y, which indicates whether a client subscribed to a term deposit. Having a large dataset allows us to run models more effectively.

The summary provides central tendency (mean, median) and spread (quartiles, min, max) for numeric variables, and frequency counts for categorical ones. This helps us quickly spot imbalances or unusual values.

First step is preprocessing:

```
# Drop duration to avoid leakage and handle pdays
bank <- bank %>%
  select(-duration) %>%
  mutate(
    pdays_flag = if_else(pdays == -1, "Never Contacted", "Contacted"),
    pdays = if_else(pdays == -1, NA_real_, pdays)
  )

# Convert all character variables to factors (required for recipes)
bank <- bank %>% mutate(across(where(is.character), as.factor))

# Make sure target variable is factor before splitting
bank <- bank %>% mutate(y = as.factor(y))

# Examine class balance
table(bank$y)
```

```
##
##    no   yes
## 39922  5289
```

Note: We removed `duration` to prevent target leakage. `pdays` is turned into NA for those never contacted and we keep a `pdays_flag` to retain signal.

Here we train-test split (80/20) similar to how we did in assignment 2.We split the dataset into **80% training** and **20% testing** while stratifying by the target variable `y` to preserve class balance.

```
set.seed(123)
split <- initial_split(bank, prop = 0.8, strata = y)
train_data <- training(split)
test_data <- testing(split)
```

We will create a recipe that:

- Imputes missing numeric values (e.g., `pdays`) with median.
- Removes near-zero variance predictors.
- One-hot encodes categorical variables.
- Normalizes numeric predictors — especially useful for SVM.

```
rec <- recipe(y ~ ., data = train_data) %>%
  step_impute_median(all_numeric(), -all_outcomes()) %>%
  step_nzv(all_predictors()) %>%
  step_dummy(all_nominal_predictors(), -all_outcomes(), one_hot = TRUE) %>%
  step_normalize(all_numeric(), -all_outcomes())
```

```r
# Define metrics (Use yardstick metrics only to avoid caret conflicts)
my_metrics <- yardstick::metric_set(
  yardstick::accuracy,
  yardstick::precision,
  yardstick::recall,
  yardstick::f_meas
)
```

A decision tree with reproduce baseline

```r
dt_spec <- decision_tree(mode = "classification") %>%
  set_engine("rpart")

dt_wflow <- workflow() %>%
  add_model(dt_spec) %>%
  add_recipe(rec)

# Fit the decision tree
dt_fit <- fit(dt_wflow, data = train_data)

# Predict on test set
dt_pred_prob <- predict(dt_fit, test_data, type = "prob")
dt_pred_class <- predict(dt_fit, test_data)

# Combine predictions with actuals
dt_results <- bind_cols(test_data %>% select(y), dt_pred_class, dt_pred_prob)
names(dt_results)[2] <- ".pred_class"

# Evaluate model
dt_metrics <- my_metrics(dt_results, truth = y, estimate = .pred_class)
dt_cm <- conf_mat(dt_results, truth = y, estimate = .pred_class)

# Output results
dt_metrics
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  binary         0.893
## 2 precision binary         0.900
## 3 recall    binary         0.988
## 4 f_meas    binary         0.942
```

```r
dt_cm
```

```
##           Truth
## Prediction   no  yes
##        no  7887  873
##        yes   98  185
```

The baseline Decision Tree gives us an interpretable model. We evaluate its performance using accuracy, precision, recall, and F1-score. The confusion matrix shows how well it distinguishes between “yes” and

"no". The baseline decision tree achieved an accuracy of 0.893, with strong recall (0.988) but slightly lower precision (0.900), indicating that the model correctly identifies most positive cases but occasionally misclassifies negatives as positives. Overall, it provides a simple yet effective starting point for classification.

Decision Tree (depth = 3, minsplit = 2) — tuned version from Assignment 2

```
dt_spec2 <- decision_tree(mode = "classification") %>%
set_engine("rpart", control = rpart.control(maxdepth = 3, minsplit = 2, cp = 0))

dt_wflow2 <- workflow() %>% add_model(dt_spec2) %>% add_recipe(rec)

dt_fit2 <- fit(dt_wflow2, data = train_data)

dt2_pred_class <- predict(dt_fit2, test_data)
dt2_pred_prob <- predict(dt_fit2, test_data, type = "prob")

dt2_results <- bind_cols(test_data %>% select(y), dt2_pred_class, dt2_pred_prob)
names(dt2_results)[2] <- ".pred_class"

dt2_metrics <- my_metrics(dt2_results, truth = y, estimate = .pred_class)
dt2_cm <- conf_mat(dt2_results, truth = y, estimate = .pred_class)
dt2_metrics
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  binary         0.892
## 2 precision binary         0.900
## 3 recall    binary         0.988
## 4 f_meas    binary         0.942
```

```
dt2_cm
```

```
##           Truth
## Prediction   no  yes
##        no  7886  879
##        yes   99  179
```

We constrain the tree to **depth 3** to prevent overfitting and ensure interpretability. This often improves generalization while maintaining good accuracy. Tuning the tree with a maximum depth of 3 produced nearly identical performance (accuracy = 0.892), suggesting limited sensitivity to parameter changes. This reflects the model's simplicity and its tendency to underfit complex patterns.

Random Forest baseline (reproduce)

```
rf_spec <- rand_forest(mode = "classification") %>%
set_engine("ranger")

rf_wflow <- workflow() %>% add_model(rf_spec) %>% add_recipe(rec)

rf_fit <- fit(rf_wflow, data = train_data)

rf_pred_class <- predict(rf_fit, test_data)
```

```r
rf_pred_prob <- predict(rf_fit, test_data, type = "prob")

rf_results <- bind_cols(test_data %>% select(y), rf_pred_class, rf_pred_prob)
names(rf_results)[2] <- ".pred_class"

rf_metrics <- my_metrics(rf_results, truth = y, estimate = .pred_class)
rf_cm <- conf_mat(rf_results, truth = y, estimate = .pred_class)
rf_metrics
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  binary         0.894
## 2 precision binary         0.903
## 3 recall    binary         0.986
## 4 f_meas    binary         0.943
```

```r
rf_cm
```

```
##           Truth
## Prediction   no  yes
##        no  7872  845
##        yes  113  213
```

Random Forest aggregates multiple decision trees, improving predictive power and reducing variance.

The baseline random forest slightly improved accuracy to 0.894, with balanced precision (0.903) and recall (0.986). The ensemble approach reduced variance compared to single trees, producing more stable predictions.

Random Forest tuned (trees = 500, mtry = 4)

```r
rf_spec_tuned <- rand_forest(mode = "classification", trees = 500, mtry = 4) %>%
set_engine("ranger")

rf_wflow_tuned <- workflow() %>% add_model(rf_spec_tuned) %>% add_recipe(rec)
rf_fit_tuned <- fit(rf_wflow_tuned, data = train_data)

rf_tuned_pred_class <- predict(rf_fit_tuned, test_data)
rf_tuned_pred_prob <- predict(rf_fit_tuned, test_data, type = "prob")

rf_tuned_results <- bind_cols(test_data %>% select(y), rf_tuned_pred_class, rf_tuned_pred_prob)
names(rf_tuned_results)[2] <- ".pred_class"

rf_tuned_metrics <- my_metrics(rf_tuned_results, truth = y, estimate = .pred_class)
rf_tuned_cm <- conf_mat(rf_tuned_results, truth = y, estimate = .pred_class)
rf_tuned_metrics
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  binary         0.892
## 2 precision binary         0.899
```

```
## 3 recall    binary         0.990
## 4 f_meas    binary         0.942
```

```
rf_tuned_cm
```

```
##           Truth
## Prediction   no  yes
##        no  7903  891
##        yes   82  167
```

Increasing the number of trees improves stability, and adjusting `mtry` controls how features are sampled per split.

Tuning the random forest (500 trees, mtry = 4) maintained similar accuracy (0.893) with marginally better recall. This suggests the baseline configuration was already near optimal, and performance gains from tuning were minimal.

Support Vector Machine (RBF) — tuning and evaluation

We'll tune cost (C) and rbf_sigma (the RBF kernel width). We use a small grid so it runs in reasonable time in class.

```
# --- Optimized SVM (RBF) with faster tuning ---
svm_spec <- svm_rbf(
  mode = "classification",
  cost = tune(),
  rbf_sigma = tune()
) %>%
  set_engine("kernlab")

svm_wflow <- workflow() %>%
  add_model(svm_spec) %>%
  add_recipe(rec)

# --- Sample only 40% of training data for faster tuning ---
set.seed(456)
train_sample <- train_data %>% sample_frac(0.4)

# --- 2-fold CV for speed (instead of 3) ---
cv_folds <- vfold_cv(train_sample, v = 2, strata = y)

# --- Smaller grid: 2x2 combinations ---
svm_grid <- grid_regular(
  cost(range = c(-1, 1)),        # 10^-1 to 10^1
  rbf_sigma(range = c(-3, -1)),  # 10^-3 to 10^-1
  levels = c(cost = 2, rbf_sigma = 2)
)

# --- Tune ---
set.seed(345)
svm_tune_res <- tune_grid(
  svm_wflow,
  resamples = cv_folds,
```

```
  grid = svm_grid,
  metrics = metric_set(accuracy),
  control = control_grid(save_pred = TRUE)
)
```

```
## line search fails -1.79547 0.5920914 4.530361e-05 -3.547158e-05 -1.647802e-07 -7.187985e-08 -4.91544
```

```
# --- Best parameters by accuracy ---
best_svm <- select_best(svm_tune_res, metric = "accuracy")
best_svm
```

```
## # A tibble: 1 x 3
##    cost rbf_sigma .config
##   <dbl>     <dbl> <chr>
## 1   0.5     0.001 pre0_mod1_post0
```

```
# --- Finalize and refit on full training data ---
svm_final <- finalize_workflow(svm_wflow, best_svm) %>%
  fit(data = train_data)

# --- Predict and evaluate ---
svm_pred_class <- predict(svm_final, test_data)
svm_pred_prob <- predict(svm_final, test_data, type = "prob")

svm_results <- bind_cols(test_data %>% select(y), svm_pred_class, svm_pred_prob)
names(svm_results)[2] <- ".pred_class"

svm_metrics <- my_metrics(svm_results, truth = y, estimate = .pred_class)
svm_cm <- conf_mat(svm_results, truth = y, estimate = .pred_class)

svm_metrics
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  binary         0.893
## 2 precision binary         0.900
## 3 recall    binary         0.988
## 4 f_meas    binary         0.942
```

```
svm_cm
```

```
##           Truth
## Prediction   no  yes
##        no  7887  873
##        yes   98  185
```

The optimized SVM model achieved an **accuracy of 0.893**, comparable to the tree-based models. Despite requiring higher computational effort, its precision (**0.900**) and recall (**0.988**) show strong classification balance. Runtime was reduced by subsampling and simplifying the grid search, preserving accuracy while cutting training time dramatically.

Note: The original SVM tuning setup (3-fold CV with a 3×3 grid) was computationally intensive due to the large dataset size. To reduce runtime, the tuning process was adjusted by subsampling 40% of the training data, using 2-fold cross-validation, and simplifying the grid to two levels per parameter. This reduced model runs from 27 to 8 and cut runtime from 30+ minutes to about 5 minutes while maintaining similar accuracy for comparison.

```
#SVM - Linear kernel (comparison)
#Linear SVM (fast) - single tuning parameter (cost)

svm_lin_spec <- svm_linear(mode = "classification", cost = tune()) %>%
set_engine("kernlab")

svm_lin_wflow <- workflow() %>% add_model(svm_lin_spec) %>% add_recipe(rec)

#Quick 2x grid for cost on a small subsample or full (fast)

set.seed(789)
train_sample2 <- train_data %>% sample_frac(0.4)
cv_folds2 <- vfold_cv(train_sample2, v = 2, strata = y)

svm_lin_grid <- grid_regular(cost(range = c(-1, 1)), levels = 2)

svm_lin_tune <- tune_grid(
svm_lin_wflow,
resamples = cv_folds2,
grid = svm_lin_grid,
metrics = metric_set(accuracy),
control = control_grid(save_pred = TRUE)
)
```

```
## line search fails -1.421379 0.790393 2.386366e-05 2.386347e-05 -2.097031e-07 -2.097126e-07 -1.000875
```

```
best_svm_lin <- select_best(svm_lin_tune, metric = "accuracy")
best_svm_lin
```

```
## # A tibble: 1 x 2
##    cost .config
##   <dbl> <chr>
## 1   0.5 pre0_mod1_post0
```

```
svm_lin_final <- finalize_workflow(svm_lin_wflow, best_svm_lin) %>% fit(data = train_data)

svm_lin_pred_class <- predict(svm_lin_final, test_data)
svm_lin_pred_prob <- predict(svm_lin_final, test_data, type = "prob")

svm_lin_results <- bind_cols(test_data %>% select(y), svm_lin_pred_class, svm_lin_pred_prob)
names(svm_lin_results)[2] <- ".pred_class"

svm_lin_metrics <- my_metrics(svm_lin_results, truth = y, estimate = .pred_class)
svm_lin_cm <- conf_mat(svm_lin_results, truth = y, estimate = .pred_class)

svm_lin_metrics
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  binary         0.892
## 2 precision binary         0.900
## 3 recall    binary         0.987
## 4 f_meas    binary         0.942
```

```
svm_lin_cm
```

```
##           Truth
## Prediction   no  yes
##        no  7885  873
##        yes  100  185
```

```
compare_tbl <- tibble(
  model = c("DT_baseline", "DT_tuned", "RF_baseline", "RF_tuned", "SVM_rbf"),
  accuracy = c(
    dt_metrics %>% filter(.metric == "accuracy") %>% pull(.estimate),
    dt2_metrics %>% filter(.metric == "accuracy") %>% pull(.estimate),
    rf_metrics %>% filter(.metric == "accuracy") %>% pull(.estimate),
    rf_tuned_metrics %>% filter(.metric == "accuracy") %>% pull(.estimate),
    svm_metrics %>% filter(.metric == "accuracy") %>% pull(.estimate)
  )
)

compare_tbl
```

```
## # A tibble: 5 x 2
##   model       accuracy
##   <chr>          <dbl>
## 1 DT_baseline    0.893
## 2 DT_tuned       0.892
## 3 RF_baseline    0.894
## 4 RF_tuned       0.892
## 5 SVM_rbf        0.893
```

We tune two hyperparameters:

- `cost` (penalty for misclassification)

- `rbf_sigma` (controls the kernel width).

We use 3-fold CV and a small grid for efficiency.

All models performed similarly, with accuracy values clustering around **0.892–0.894**. The random forest slightly outperformed others in raw accuracy, but the SVM and decision tree models demonstrated comparable results, highlighting that model choice may hinge more on interpretability and computational efficiency than predictive gain.

Notes on Performance and Reproducibility

- SVM tuning is computationally expensive — we used 3-fold CV and small parameter grids to reduce runtime.

- Random Forests generally outperform Decision Trees in accuracy but lose interpretability.
- All results are reproducible with `set.seed()` for splits and resampling.