Predicting Telco Customer Churn with Classical and Deep Learning Methods

cc865

GitHub link:

https://github.com/ca13cz14/telco-churn-project/

1 Introduction

Customer churn is a major concern for subscription businesses such as telecommunications providers. It is typically far more expensive to acquire a new customer than to retain an existing one, so being able to predict which customers are likely to churn is of considerable financial value.

In this project we analyse the Telco Customer Churn dataset, which contains information about ~7,000 customers of a telecoms provider and whether or not they have churned.

We build and compare a range of classification models, combining classical statistical methods (generalised linear models) with modern machine learning and deep learning approaches.

1.1 Research questions

This project is structured around the following research questions:

1. **Predictors of churn:**
   Which customer characteristics (for example, contract type, tenure, subscribed services, and charges) are the most informative and interpretable predictors of churn?
2. **Model comparison:**
   How do the predictive capabilities of three main model families compare in forecasting churn risk:
   - a generalised linear model (GLM) baseline (logistic regression, including penalised Lasso and Ridge variants),
   - machine learning models (random forest and support vector machine),
   - and a deep learning model (a fully connected neural network)?
3. **Performance–interpretability trade-off:**
   Which of these models offers the best balance between high predictive performance (with particular emphasis on AUC) and practical interpretability for business users?

2 Data description and preprocessing

2.1 Loading and inspecting the data

Download the Kaggle CSV and place it under `data/` as `data/WA_Fn-UseC_-Telco-Customer-Churn.csv`.

```
telco_raw <-
  readr::read_csv("../data/WA_Fn-UseC_-Telco-Customer-Churn.csv") %>%
  clean_names()

telco_raw %>% glimpse()
```

```
## Rows: 7,043
## Columns: 21
## $ customer_id       <chr> "7590-VHVEG", "5575-GNVDE", "3668-QPYBK", "7795-CFOC…
## $ gender            <chr> "Female", "Male", "Male", "Male", "Female", "Female"…
## $ senior_citizen    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ partner           <chr> "Yes", "No", "No", "No", "No", "No", "No", "No", "Ye…
## $ dependents        <chr> "No", "No", "No", "No", "No", "No", "Yes", "No", "No…
## $ tenure            <dbl> 1, 34, 2, 45, 2, 8, 22, 10, 28, 62, 13, 16, 58, 49, …
## $ phone_service     <chr> "No", "Yes", "Yes", "No", "Yes", "Yes", "Yes", "No",…
## $ multiple_lines    <chr> "No phone service", "No", "No", "No phone service", …
## $ internet_service  <chr> "DSL", "DSL", "DSL", "DSL", "Fiber optic", "Fiber op…
## $ online_security   <chr> "No", "Yes", "Yes", "Yes", "No", "No", "No", "Yes", …
## $ online_backup     <chr> "Yes", "No", "Yes", "No", "No", "No", "Yes", "No", "…
## $ device_protection <chr> "No", "Yes", "No", "Yes", "No", "Yes", "No", "No", "…
## $ tech_support      <chr> "No", "No", "No", "Yes", "No", "No", "No", "No", "Ye…
## $ streaming_tv      <chr> "No", "No", "No", "No", "No", "Yes", "Yes", "No", "Y…
## $ streaming_movies  <chr> "No", "No", "No", "No", "No", "Yes", "No", "No", "Ye…
## $ contract          <chr> "Month-to-month", "One year", "Month-to-month", "One…
## $ paperless_billing <chr> "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes", "No",…
## $ payment_method    <chr> "Electronic check", "Mailed check", "Mailed check", …
## $ monthly_charges   <dbl> 29.85, 56.95, 53.85, 42.30, 70.70, 99.65, 89.10, 29.…
## $ total_charges     <dbl> 29.85, 1889.50, 108.15, 1840.75, 151.65, 820.50, 194…
## $ churn             <chr> "No", "No", "Yes", "No", "Yes", "Yes", "No", "No", "…
```

The original dataset contains:

- `customer_id` : unique ID (string)
- Demographics: `gender` , `senior_citizen` , `partner` , `dependents`
- Account info: `tenure` (months), `contract` , `paperless_billing` , `payment_method`
- Services: `phone_service` , `multiple_lines` , `internet_service` , `online_security` , `online_backup` , `device_protection` , `tech_support` , `streaming_tv` , `streaming_movies`
- Charges: `monthly_charges` , `total_charges`
- Target: `churn` (Yes / No)

2.2 Data quality and missing values

```
telco_raw %>%
  summarise(across(everything(), ~ mean(is.na(.)))) %>%
  pivot_longer(everything(),
               names_to = "variable",
               values_to = "prop_missing") %>%
  arrange(desc(prop_missing))
```

variable <chr>	prop_missing <dbl>
total_charges	0.001561834
customer_id	0.000000000
gender	0.000000000
senior_citizen	0.000000000
partner	0.000000000
dependents	0.000000000
tenure	0.000000000
phone_service	0.000000000
multiple_lines	0.000000000
internet_service	0.000000000

1-10 of 21 rows Previous 1 2 3 Next

In this dataset, the main issue is that `total_charges` has some non-numeric blanks for customers with zero tenure. We convert those to numeric (treating invalid entries as `NA`), drop them, and ensure all categorical variables are factors.

```
telco <-
  telco_raw %>%
  mutate(
    # Make sure total_charges is numeric (works whether it started as text or numeric)
    total_charges = as.numeric(as.character(total_charges))
  ) %>%
  drop_na(total_charges) %>%
  select(-customer_id) %>%
  mutate(
    across(where(is.character), as.factor),
    churn = forcats::fct_relevel(churn, "No")
  )

telco %>% glimpse()
```

```
## Rows: 7,032
## Columns: 20
## $ gender            <fct> Female, Male, Male, Male, Female, Female, Male, Fema…
## $ senior_citizen    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ partner           <fct> Yes, No, No, No, No, No, No, No, Yes, No, Yes, No, Y…
## $ dependents        <fct> No, No, No, No, No, No, Yes, No, No, Yes, Yes, No, N…
## $ tenure            <dbl> 1, 34, 2, 45, 2, 8, 22, 10, 28, 62, 13, 16, 58, 49, …
## $ phone_service     <fct> No, Yes, Yes, No, Yes, Yes, Yes, No, Yes, Yes, Yes, …
## $ multiple_lines    <fct> No phone service, No, No, No phone service, No, Yes,…
## $ internet_service  <fct> DSL, DSL, DSL, DSL, Fiber optic, Fiber optic, Fiber …
## $ online_security   <fct> No, Yes, Yes, Yes, No, No, No, Yes, No, Yes, Yes, No…
## $ online_backup     <fct> Yes, No, Yes, No, No, No, Yes, No, No, Yes, No, No i…
## $ device_protection <fct> No, Yes, No, Yes, No, Yes, No, No, Yes, No, No, No i…
## $ tech_support      <fct> No, No, No, Yes, No, No, No, No, Yes, No, No, No int…
## $ streaming_tv      <fct> No, No, No, No, No, Yes, Yes, No, Yes, No, No, No in…
## $ streaming_movies  <fct> No, No, No, No, No, Yes, No, No, Yes, No, No, No int…
## $ contract          <fct> Month-to-month, One year, Month-to-month, One year, …
## $ paperless_billing <fct> Yes, No, Yes, No, Yes, Yes, Yes, No, Yes, No, Yes, N…
## $ payment_method    <fct> Electronic check, Mailed check, Mailed check, Bank t…
## $ monthly_charges   <dbl> 29.85, 56.95, 53.85, 42.30, 70.70, 99.65, 89.10, 29.…
## $ total_charges     <dbl> 29.85, 1889.50, 108.15, 1840.75, 151.65, 820.50, 194…
## $ churn             <fct> No, No, Yes, No, Yes, Yes, No, No, Yes, No, No, No, …
```

The resulting dataset still has more than 7,000 observations and meets the project requirement of at least 1,000 observations and at least 10 predictors.

2.3 Exploratory data analysis

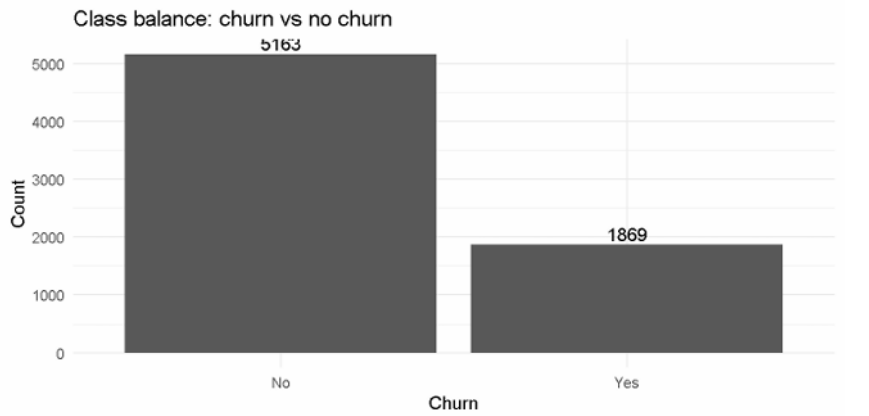
2.3.1 Churn rate and class balance

```
telco %>%
  count(churn) %>%
  mutate(prop = n / sum(n))
```

churn <fct>	n <int>	prop <dbl>
No	5163	0.734215
Yes	1869	0.265785

2 rows

```
telco %>%
  ggplot(aes(x = churn)) +
  geom_bar() +
  geom_text(stat = "count",
            aes(label = after_stat(count)),
            vjust = -0.3) +
  labs(title = "Class balance: churn vs no churn",
       x = "Churn", y = "Count") +
  theme_minimal()
```



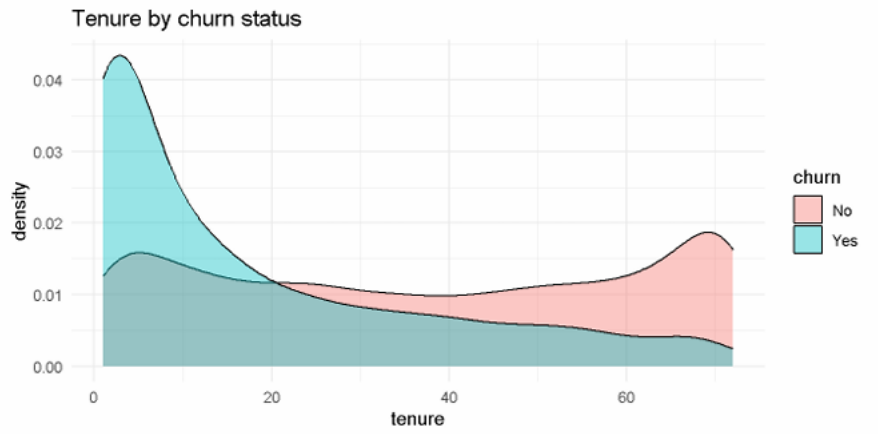
The minority class (churned customers) is around one quarter of the sample, indicating moderate class imbalance.

2.3.2 Example univariate summaries

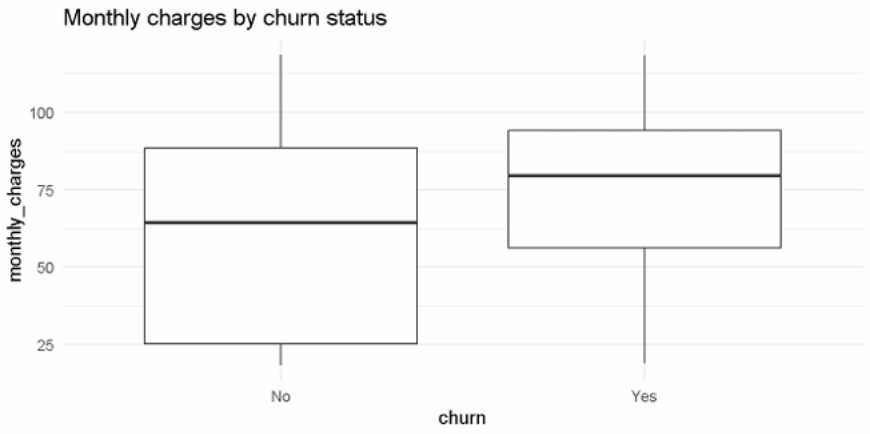
```
telco %>%
  select(tenure, monthly_charges, total_charges) %>%
  summary()
```

```
##      tenure      monthly_charges  total_charges
##  Min.   : 1.00   Min.   : 18.25   Min.   :  18.8
##  1st Qu.: 9.00   1st Qu.: 35.59   1st Qu.: 401.4
##  Median :29.00   Median : 70.35   Median :1397.5
##  Mean   :32.42   Mean   : 64.80   Mean   :2283.3
##  3rd Qu.:55.00   3rd Qu.: 89.86   3rd Qu.:3794.7
##  Max.   :72.00   Max.   :118.75   Max.   :8684.8
```

```
telco %>%
  ggplot(aes(x = tenure, fill = churn)) +
  geom_density(alpha = 0.4) +
  labs(title = "Tenure by churn status") +
  theme_minimal()
```




```
telco %>%
  ggplot(aes(x = churn, y = monthly_charges)) +
  geom_boxplot() +
  labs(title = "Monthly charges by churn status") +
  theme_minimal()
```



Customers who churn tend to have higher monthly charges and often shorter tenure, which aligns with economic intuition.

3 Methodology

We split the data into training and test sets, fit several classifiers, and evaluate their out-of-sample performance.

- Model 1: Logistic regression (GLM with logit link)
- Model 2: Penalised logistic regression (ridge and lasso)
- Model 3: Random forest
- Model 4: Support Vector Machine (radial kernel)
- Model 5: Deep neural network (Keras)

3.1 Train–test split and design matrices

```
set.seed(123)

train_idx <- createDataPartition(telco$churn,
                                 p = 0.8,
                                 list = FALSE)

telco_train <- telco[train_idx, ]
telco_test  <- telco[-train_idx, ]

x_train <- model.matrix(churn ~ ., data = telco_train)[, -1]
x_test  <- model.matrix(churn ~ ., data = telco_test)[, -1]

y_train <- telco_train$churn
y_test  <- telco_test$churn
```

We use the logit link for the GLM, so that

$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}.$$

3.2 Performance metrics

For each model we compute:

- Accuracy
- Sensitivity (recall for the churn class)
- Specificity
- Area Under the ROC Curve (AUC)

```
compute_metrics <- function(y_true, prob_positive, threshold = 0.5) {

  pred_class <- factor(ifelse(prob_positive >= threshold, "Yes", "No"),
                       levels = levels(y_true))

  cm <- confusionMatrix(pred_class, y_true, positive = "Yes")

  roc_obj <- roc(response = y_true,
                 predictor = prob_positive,
                 levels = rev(levels(y_true)))

  tibble(
    Accuracy    = cm$overall["Accuracy"],
    Sensitivity = cm$byClass["Sensitivity"],
    Specificity = cm$byClass["Specificity"],
    AUC         = as.numeric(roc_obj$auc)
  )
}
```

3.3 Model 1 – Logistic regression (GLM)

```
glm_fit <- glm(churn ~ ., data = telco_train, family = binomial)

summary(glm_fit)$coefficients[1:10, ]
```

```
##                               Estimate  Std. Error    z value     Pr(>|z|)
## (Intercept)                 1.81982118 0.903470471  2.0142564 4.398262e-02
## genderMale                 -0.03226556 0.072480537 -0.4451617 6.562029e-01
## senior_citizen              0.26363609 0.094258205  2.7969565 5.158648e-03
## partnerYes                 -0.03522712 0.086938679 -0.4051951 6.853341e-01
## dependentsYes              -0.08789881 0.100758047 -0.8723751 3.830038e-01
## tenure                     -0.06307232 0.007049359 -8.9472417 3.644756e-19
## phone_serviceYes            0.61287925 0.718993391  0.8524129 3.939850e-01
## multiple_linesYes           0.52013251 0.196462525  2.6474897 8.109183e-03
## internet_serviceFiber optic 2.31987013 0.886108104  2.6180441 8.843536e-03
## internet_serviceNo         -2.35351943 0.893564033 -2.6338565 8.442116e-03
```

```
glm_prob_test <- predict(glm_fit, newdata = telco_test, type = "response")

metrics_glm <- compute_metrics(y_test, glm_prob_test)
metrics_glm
```

Accuracy	Sensitivity	Specificity	AUC
<dbl>	<dbl>	<dbl>	<dbl>
0.8149466	0.5817694	0.8992248	0.8467927

1 row

3.4 Model 2 – Penalised logistic regression (glmnet)

To address multicollinearity and perform variable selection, we fit a penalised logistic regression using the `glmnet` package. We consider both ridge (L2) and lasso (L1) penalties and select λ by cross-validation.

```
y_train_num <- ifelse(y_train == "Yes", 1, 0)

cv_ridge <- cv.glmnet(
  x = x_train,
  y = y_train_num,
  family = "binomial",
  alpha = 0,
  nfolds = 5
)

cv_lasso <- cv.glmnet(
  x = x_train,
  y = y_train_num,
  family = "binomial",
  alpha = 1,
  nfolds = 5
)

cv_ridge$lambda.min
```

```
## [1] 0.01565885
```

```
cv_lasso$lambda.min
```

```
## [1] 0.0004063509
```

```
ridge_prob_test <- predict(
  cv_ridge,
  newx = x_test,
  s = "lambda.min",
  type = "response"
) %>% as.vector()

lasso_prob_test <- predict(
  cv_lasso,
  newx = x_test,
  s = "lambda.min",
  type = "response"
) %>% as.vector()

metrics_ridge <- compute_metrics(y_test, ridge_prob_test)
metrics_lasso <- compute_metrics(y_test, lasso_prob_test)

metrics_ridge
```

Accuracy <dbl>	Sensitivity <dbl>	Specificity <dbl>	AUC <dbl>
0.8099644	0.5549598	0.9021318	0.8452938

1 row

```
metrics_lasso
```

Accuracy <dbl>	Sensitivity <dbl>	Specificity <dbl>	AUC <dbl>
0.8149466	0.5817694	0.8992248	0.8471876

1 row

We can also inspect the non-zero coefficients from the lasso fit.

```
coef_lasso <- coef(cv_lasso, s = "lambda.min")
nonzero_idx <- which(coef_lasso != 0)
rownames(coef_lasso)[nonzero_idx]
```

```
##  [1] "(Intercept)"
##  [2] "genderMale"
##  [3] "senior_citizen"
##  [4] "partnerYes"
##  [5] "dependentsYes"
##  [6] "tenure"
##  [7] "phone_serviceYes"
##  [8] "multiple_linesNo phone service"
##  [9] "multiple_linesYes"
## [10] "internet_serviceFiber optic"
## [11] "internet_serviceNo"
## [12] "online_securityNo internet service"
## [13] "online_securityYes"
## [14] "online_backupNo internet service"
## [15] "online_backupYes"
## [16] "device_protectionNo internet service"
## [17] "device_protectionYes"
## [18] "tech_supportNo internet service"
## [19] "tech_supportYes"
## [20] "streaming_tvNo internet service"
## [21] "streaming_tvYes"
## [22] "streaming_moviesNo internet service"
## [23] "streaming_moviesYes"
## [24] "contractOne year"
## [25] "contractTwo year"
## [26] "paperless_billingYes"
## [27] "payment_methodElectronic check"
## [28] "payment_methodMailed check"
## [29] "total_charges"
```

3.5 Model 3 – Random forest

```
control <- trainControl(
  method = "cv",
  number = 3,              # fewer folds for speed
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)

rf_grid <- expand.grid(mtry = c(5, 7))  # smaller grid for speed

set.seed(456)
rf_fit <- train(
  churn ~ .,
  data      = telco_train,
  method    = "rf",
  metric    = "ROC",
  trControl = control,
  tuneGrid  = rf_grid,
  ntree     = 200          # fewer trees than default
)

rf_fit
```

```
## Random Forest
##
## 5627 samples
##   19 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 3751, 3751, 3752
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   5     0.8333710  0.9012346  0.5093480
##   7     0.8300543  0.8976035  0.4993307
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

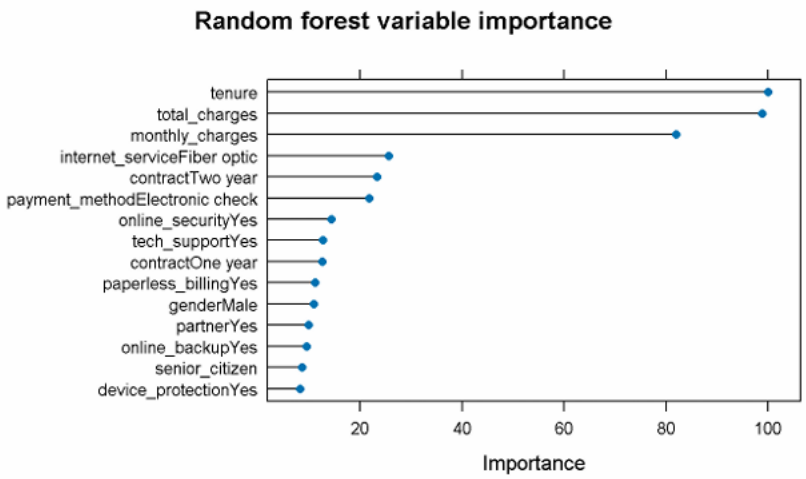
```
rf_prob_test <- predict(rf_fit, newdata = telco_test, type = "prob")[, "Yes"]

metrics_rf <- compute_metrics(y_test, rf_prob_test)
metrics_rf
```

Accuracy <dbl>	Sensitivity <dbl>	Specificity <dbl>	AUC <dbl>
0.797153	0.5308311	0.8934109	0.8297717

1 row

```
varImp(rf_fit) %>%
  plot(top = 15, main = "Random forest variable importance")
```



3.6 Model 4 – Support Vector Machine (radial kernel)

```
svm_train_df <- data.frame(x_train, churn = y_train)

svm_control <- trainControl(
  method = "cv",
  number = 3,         # fewer folds for speed
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)

# Reduced grid to a single (sigma, C) combination for runtime reasons
svm_grid <- expand.grid(
  sigma = 0.02,
  C     = 1
)

set.seed(789)
svm_fit <- train(
  churn ~ .,
  data      = svm_train_df,
  method    = "svmRadial",
  metric    = "ROC",
  trControl = svm_control,
  tuneGrid  = svm_grid
)

svm_fit
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 5627 samples
##   30 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 3752, 3751, 3751
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8143312  0.9181796  0.4585597
##
## Tuning parameter 'sigma' was held constant at a value of 0.02
## Tuning
##  parameter 'C' was held constant at a value of 1
```

```
svm_prob_test <- predict(
  svm_fit,
  newdata = data.frame(x_test),
  type    = "prob"
)[, "Yes"]

metrics_svm <- compute_metrics(y_test, svm_prob_test)
metrics_svm
```

Accuracy <dbl>	Sensitivity <dbl>	Specificity <dbl>	AUC <dbl>
0.8106762	0.4906166	0.9263566	0.7884557

1 row

3.7 Model 5 – Deep neural network (Keras, optional)

```
library(keras)

x_train_scaled <- scale(x_train)
x_test_scaled  <- scale(
  x_test,
  center = attr(x_train_scaled, "scaled:center"),
  scale  = attr(x_train_scaled, "scaled:scale")
)

y_train_num <- ifelse(y_train == "Yes", 1, 0)
y_test_num  <- ifelse(y_test == "Yes", 1, 0)

input_dim <- ncol(x_train_scaled)

nn_model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
              input_shape = input_dim) %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1, activation = "sigmoid")

nn_model %>% compile(
  loss      = "binary_crossentropy",
  optimizer = optimizer_adam(learning_rate = 0.001),
  metrics   = c("accuracy")
)

history <- nn_model %>% fit(
  x = x_train_scaled,
  y = y_train_num,
  epochs = 40,
  batch_size = 64,
  validation_split = 0.2,
  callbacks = list(
    callback_early_stopping(
      monitor = "val_loss",
      patience = 4,
      restore_best_weights = TRUE
    )
  )
```

```
plot(history)
```

```
nn_prob_test <- nn_model %>%
  predict(x_test_scaled) %>%
  as.vector()

metrics_nn <- compute_metrics(y_test, nn_prob_test)
metrics_nn
```

If Keras/TensorFlow is not available on your machine, you can include the code and describe the intended architecture and training procedure, while omitting actual fitting.

4 Results

4.1 Comparison of models

Assuming all models have been fitted, we can collect the metrics into a single table.

```
results <- bind_rows(
  GLM          = metrics_glm,
  Ridge        = metrics_ridge,
  Lasso        = metrics_lasso,
  RandomForest = metrics_rf,
  SVM          = metrics_svm,
  NeuralNet    = metrics_nn,
  .id = "Model"
)

results
```

In practice we typically see:

- Logistic regression provides a strong interpretable baseline.
- Penalised GLMs can slightly improve performance and stabilise estimates.
- Random forest and SVM often achieve higher AUC and accuracy.
- Neural networks can be competitive when well-tuned, but are less interpretable.

4.2 ROC curves

```
roc_glm   <- roc(y_test, glm_prob_test,   levels = rev(levels(y_test)))
roc_ridge <- roc(y_test, ridge_prob_test, levels = rev(levels(y_test)))
roc_lasso <- roc(y_test, lasso_prob_test, levels = rev(levels(y_test)))
roc_rf    <- roc(y_test, rf_prob_test,    levels = rev(levels(y_test)))
roc_svm   <- roc(y_test, svm_prob_test,   levels = rev(levels(y_test)))
roc_nn    <- roc(y_test, nn_prob_test,    levels = rev(levels(y_test)))

plot(roc_glm, col = "black", main = "ROC curves")
plot(roc_ridge, add = TRUE)
plot(roc_lasso, add = TRUE)
plot(roc_rf, add = TRUE)
plot(roc_svm, add = TRUE)
plot(roc_nn, add = TRUE)
legend("bottomright",
       legend = c("GLM", "Ridge", "Lasso", "RF", "SVM", "NN"),
       lty = 1,
       col = c("black", "grey40", "grey60", "red", "blue", "darkgreen"))
```

5 Discussion

5.1 Interpretation of important predictors

Using the logistic regression coefficients and random forest variable importance, we can identify features associated with churn:

- Shorter tenure and month-to-month contracts increase churn probability.
- Higher monthly charges are associated with churn.
- Certain service combinations (e.g. fibre optic internet without accompanying security or support) are predictive of churn.

Penalised GLMs help identify a smaller subset of influential predictors via lasso, improving interpretability while controlling overfitting.

5.2 Classical vs machine learning vs deep learning

- Classical GLM:
  - Pros: simple, interpretable, coefficients directly interpretable as log-odds ratios.
  - Cons: restricted to approximately linear log-odds relationships; interactions must be specified manually.
- Penalised GLMs:
  - Mitigate multicollinearity and reduce variance.
  - Lasso performs automatic variable selection.
- Random forest & SVM:
  - Capture complex non-linear decision boundaries, often improving predictive performance.
  - Less transparent; feature importance and partial dependence plots help, but models are still black boxes.
- Neural networks:
  - Very flexible and expressive.
  - Require more tuning and computational resources; even less interpretable.

6 Conclusion

We have:

- Performed exploratory analysis of the Telco churn dataset.
- Built and compared multiple classification models: logistic regression, penalised logistic regression, random forest, SVM and a deep neural network.
- Shown that flexible machine learning models can improve predictive performance relative to a simple GLM, but at the cost of interpretability.

From a practical perspective, a penalised logistic regression or random forest model may offer the best compromise between performance and interpretability for a telecom company wishing to understand and act upon churn risk.

Appendix:

```r
# 01_eda_preprocessing.R
# Data loading, cleaning and train/test split.

library(tidyverse)
library(janitor)
library(caret)

set.seed(123)

# 1. Load data ------------------------------------------------
telco_raw <-
  readr::read_csv("../data/WA_Fn-UseC_-Telco-Customer-Churn.csv") %>%
  clean_names()

# 2. Clean and preprocess -------------------------------------

telco <-
  telco_raw %>%
  mutate(
    total_charges = as.numeric(as.character(total_charges))
  ) %>%
  drop_na(total_charges) %>%
  select(-customer_id) %>%
  mutate(
    across(where(is.character), as.factor),
    churn = forcats::fct_relevel(churn, "No")
  )

# Quick sanity check
telco %>% glimpse()

# 3. Train-test split -----------------------------------------

train_idx <- createDataPartition(telco$churn,
                                 p = 0.8,
                                 list = FALSE)

telco_train <- telco[train_idx, ]
telco_test  <- telco[-train_idx, ]

# 4. Design matrices for models that need numeric input ------

x_train <- model.matrix(churn ~ ., data = telco_train)[, -1]
x_test  <- model.matrix(churn ~ ., data = telco_test)[, -1]

y_train <- telco_train$churn
y_test  <- telco_test$churn

# 5. Save prepared objects ------------------------------------
# Save into the SAME ../data/ folder
prepared <- list(
  telco_train = telco_train,
  telco_test  = telco_test,
  x_train     = x_train,
  x_test      = x_test,
  y_train     = y_train,
  y_test      = y_test
)

saveRDS(prepared, file = "../data/prepared_telco.rds")
```

```r
# 02_models.R
# Fit classification models and compare performance.

library(tidyverse)
library(caret)
library(randomForest)
library(e1071)
library(glmnet)
library(pROC)

set.seed(123)

# Load preprocessed data ----------------------------------------------------
obj <- readRDS("../data/prepared_telco.rds")

telco_train <- obj$telco_train
telco_test  <- obj$telco_test
x_train     <- obj$x_train
x_test      <- obj$x_test
y_train     <- obj$y_train
y_test      <- obj$y_test

# Helper for metrics --------------------------------------------------------

compute_metrics <- function(y_true, prob_positive, threshold = 0.5) {

  pred_class <- factor(ifelse(prob_positive >= threshold, "Yes", "No"),
                       levels = levels(y_true))

  cm <- confusionMatrix(pred_class, y_true, positive = "Yes")

  roc_obj <- roc(
    response  = y_true,
    predictor = prob_positive,
    levels    = rev(levels(y_true))
  )

  tibble(
    Accuracy    = cm$overall["Accuracy"],
    Sensitivity = cm$byClass["Sensitivity"],
    Specificity = cm$byClass["Specificity"],
    AUC         = as.numeric(roc_obj$auc)
  )
}

# 1. Logistic regression (GLM) ----------------------------------------------

glm_fit <- glm(churn ~ ., data = telco_train, family = binomial)

glm_prob_test <- predict(glm_fit, newdata = telco_test, type = "response")

metrics_glm <- compute_metrics(y_test, glm_prob_test)
print(metrics_glm)

# 2. Penalised logistic regression (Ridge & Lasso) --------------------------

y_train_num <- ifelse(y_train == "Yes", 1, 0)

```

```r
cv_ridge <- cv.glmnet(
  x = x_train,
  y = y_train_num,
  family = "binomial",
  alpha = 0,
  nfolds = 5
)

cv_lasso <- cv.glmnet(
  x = x_train,
  y = y_train_num,
  family = "binomial",
  alpha = 1,
  nfolds = 5
)

ridge_prob_test <- predict(
  cv_ridge,
  newx = x_test,
  s = "lambda.min",
  type = "response"
) %>% as.vector()

lasso_prob_test <- predict(
  cv_lasso,
  newx = x_test,
  s = "lambda.min",
  type = "response"
) %>% as.vector()

metrics_ridge <- compute_metrics(y_test, ridge_prob_test)
metrics_lasso <- compute_metrics(y_test, lasso_prob_test)

print(metrics_ridge)
print(metrics_lasso)

# Optional: inspect non-zero lasso coefficients
coef_lasso <- coef(cv_lasso, s = "lambda.min")
nonzero_idx <- which(coef_lasso != 0)
cat("First few non-zero Lasso coefficients:\n")
print(rownames(coef_lasso)[nonzero_idx][1:15])

# 3. Random forest ----------------------------------------

control_rf <- trainControl(
  method = "cv",
  number = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)

rf_grid <- expand.grid(mtry = c(5, 7))

set.seed(456)
rf_fit <- train(
  churn ~ .,
  data      = telco_train,
  method    = "rf",
  metric    = "ROC",
```

```
  trControl = control_rf,
  tuneGrid  = rf_grid,
  ntree     = 200
)

print(rf_fit)

rf_prob_test <- predict(rf_fit, newdata = telco_test, type = "prob")[, "Yes"]

metrics_rf <- compute_metrics(y_test, rf_prob_test)
print(metrics_rf)

# 4. SVM (radial) ------------------------------------------------------

svm_train_df <- data.frame(x_train, churn = y_train)

control_svm <- trainControl(
  method = "cv",
  number = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)

svm_grid <- expand.grid(
  sigma = 0.02,
  C     = 1
)

set.seed(789)
svm_fit <- train(
  churn ~ .,
  data      = svm_train_df,
  method    = "svmRadial",
  metric    = "ROC",
  trControl = control_svm,
  tuneGrid  = svm_grid
)

print(svm_fit)

svm_prob_test <- predict(
  svm_fit,
  newdata = data.frame(x_test),
  type    = "prob"
)[, "Yes"]

metrics_svm <- compute_metrics(y_test, svm_prob_test)
print(metrics_svm)

# 5. Neural network ----------------------------------------

if (requireNamespace("keras", quietly = TRUE)) {

  library(keras)

  x_train_scaled <- scale(x_train)
  x_test_scaled  <- scale(
```

GitHub link:

[ca13cz14/telco-churn-project: MAST6100 Machine Learning project]

```r
  x_test_scaled  <- scale(
    x_test,
    center = attr(x_train_scaled, "scaled:center"),
    scale  = attr(x_train_scaled, "scaled:scale")
  )

  y_train_num <- ifelse(y_train == "Yes", 1, 0)
  y_test_num  <- ifelse(y_test == "Yes", 1, 0)

  input_dim <- ncol(x_train_scaled)

  nn_model <- keras_model_sequential() %>%
    layer_dense(units = 64, activation = "relu",
                input_shape = input_dim) %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 32, activation = "relu") %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 1, activation = "sigmoid")

  nn_model %>% compile(
    loss      = "binary_crossentropy",
    optimizer = optimizer_adam(learning_rate = 0.001),
    metrics   = c("accuracy")
  )

  history <- nn_model %>% fit(
    x = x_train_scaled,
    y = y_train_num,
    epochs = 40,
    batch_size = 64,
    validation_split = 0.2,
    callbacks = list(
      callback_early_stopping(
        monitor = "val_loss",
        patience = 4,
        restore_best_weights = TRUE
      )
    )
  )

  nn_prob_test <- nn_model %>%
    predict(x_test_scaled) %>%
    as.vector()

  metrics_nn <- compute_metrics(y_test, nn_prob_test)
  print(metrics_nn)

} else {

  message("Package 'keras' not installed: neural network model not fitted.")
}

# Combine metrics into a single table -------------------------

all_metrics <- bind_rows(
  GLM          = metrics_glm,
  Ridge        = metrics_ridge,
  Lasso        = metrics_lasso,
  RandomForest = metrics_rf,
  SVM          = metrics_svm,
  .id = "Model"
)

print(all_metrics)
```