ST310: Group Project

A Prediction Challenge on HMEQ Data: Predicting Clients Who Default on Their Loan

40907 | 46241

Thu/29/May

1 Introduction

The HMEQ dataset contains information about home equity loans, including borrower characteristics and loan details. The goal of this project is to predict whether a client will default on their loan based on these features. We will explore the data, build baseline models, and then apply more complex machine learning techniques to improve our predictions.

Our primary motivating question is: *which mix of these quantitative and qualitative indicators most powerfully predicts default risk*, enabling an early, transparent screening of high-risk applicants.

To illuminate this, we first implemented a **custom logistic-regression via gradient descent** which is valued for its step-by-step interpretability, but our final, more sophisticated learner is a **Random Forest classifier**, chosen because of its ability to model non-linear interactions and handle heterogeneous feature sets. While HMEQ has served as a benchmark for many "black-box" solutions, our work is unique in students' perspectives, directly contrasting a hand-crafted optimizer with complex models and in rigorously evaluating model performance across a suite of metrics to address class imbalance and reflect real-world credit-risk trade-offs. A key challenge lies in metric selection: although Accuracy is widely used, AUC-ROC may offer more actionable insights when there is class imbalance.

Our report contains the following sections:

- **Exploratory Data Analysis**: A detailed analysis of the dataset, including summary statistics and visualizations.
- **Baseline Model**: A logistic regression model to establish a baseline performance.
- **Gradient Descent Model**: A custom logistic regression model using cross entropy gradient descent.
- **Interpretable Model**: A decision tree model to provide insights into feature importance.
- **High-dim Model**: A Lasso regression model to exam interactions between features.
- **Prediction Winner Model**: We fit two models (i.e„random forest and XGBoost) to focus purely on predictive accuracy.
- **Conclusion**: A summary of findings and recommendations for future work.
- **Code Appendix**: A complete code listing for reproducibility.

2 Exploratory Data Analysis

During initial data exploration, we found that the Home Equity dataset comprises baseline and loan-performance information for **5,960** recent home-equity loans, each described by **13** variables including continuous financial metrics (loan amount, mortgage due, property value, debt-to-income ratio, credit-history age, delinquency counts) and two categorical fields: REASON (loan purpose) and JOB (employment category). Therefore, we converted the categorical variables to factors and dummy-encoded them for modeling.

The target variable, BAD, indicates whether a client defaulted on their loan (1) or not (0). We first look at its distribution to understand the class balance, which is crucial for model training:

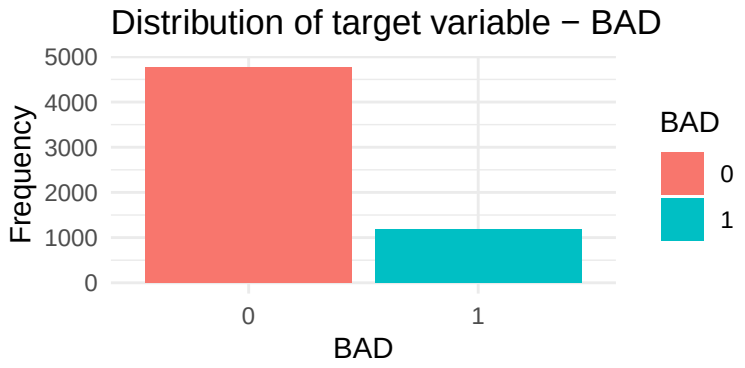

Figure 1: Distributions of target variable BAD

The plot shows a clear class imbalance: roughly 80 % of loans in the HMEQ dataset did not default (BAD = 0) while only about 20% did (BAD = 1). Such a skew means that a naïve model could achieve high accuracy simply by predicting "no default" most of the time, so we'll need to use more robust metrics (like AUC) and consider resampling or class-weighting to properly train our XGBoost model.

Now, we look at the distribution of numeric and categorical variables to understand their characteristics and potential transformations needed for modeling.

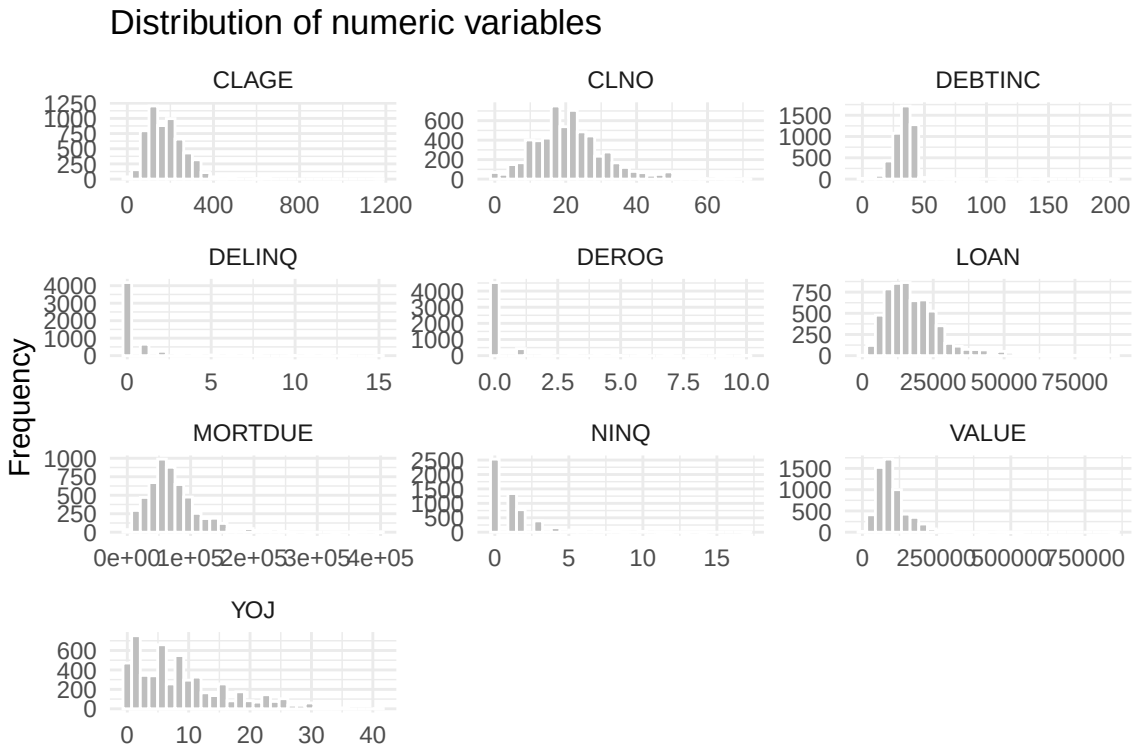

Figure 2: Distribution of all numeric variables

The histograms show that none of the numeric variables obey a normal distribution. Most features are heavily right-skewed with long tails. To address this, we will apply normalization to put all predictors on a

similar scale, and we should also consider a log transform.

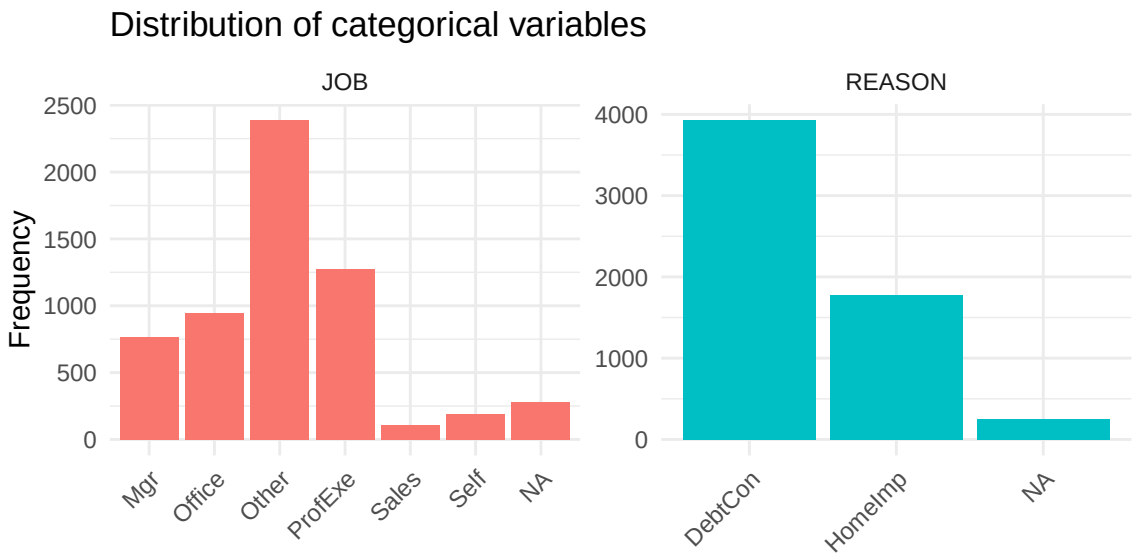

Figure 3: Distributions of categorical variables

Occupations are highly uneven: nearly 40 % of borrowers fall into the "Other" category. The loan REASON is even more concentrated: around two-thirds went to debt consolidation, about 30 % to home improvement, and only a few percent have no recorded purpose.

Now let's deal with the missing values in the dataset:

variable	NAs
BAD	0
LOAN	0
MORTDUE	518
VALUE	112
REASON	252
JOB	279
YOJ	515
DEROG	708
DELINQ	580
CLAGE	308
NINQ	510
CLNO	222
DEBTINC	1267

Oops, there're plenty of NAs in the original dataset, if we drop them directly, a large fraction of the data will be lost. We should consider the imputation for both numeric and categorical variables when fitting the models. Therefore, in later recipe building step, we will use `step_impute_mean()` for numeric variables and `step_impute_mode()` for categorical variables to handle the missing values instead of dropping them.

A correlation heatmap can help us undertand the relationships between variables, identifying potential multicollinearity issues and interactions that we might want to include in our models:

The correlation heatmap shows several relationships among the numeric variables. These observed correlations justify our decision to include interaction terms in the high-dimensional Lasso model. By adding pairwise products of related variables, we give the model a chance to capture joint effects that simple main-effect terms might miss.

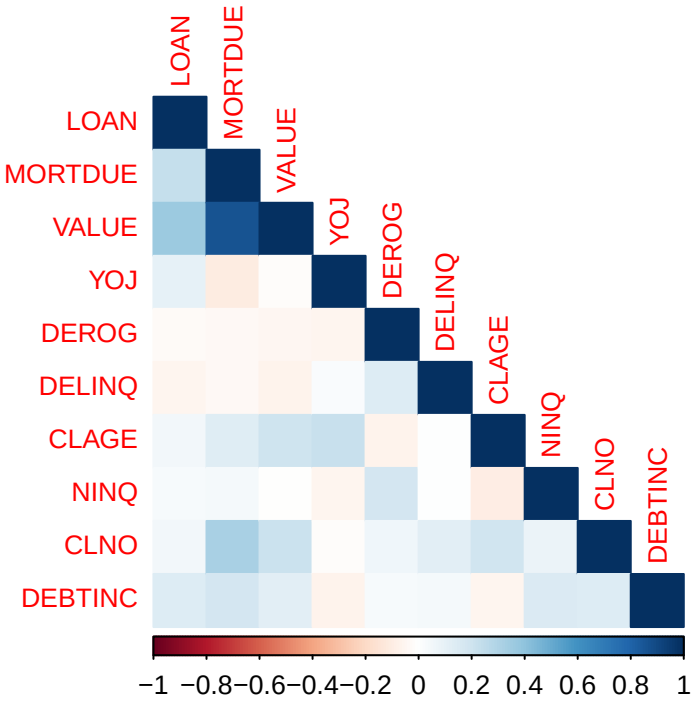

Figure 4: Heatmap of all numeric variables

3 Modeling and Evaluation

3.1 Baseline Model

We begin with a **logistic-regression** baseline. We chose it for its interpretability and natural fit to our binary "default" outcome, so that we can immediately see how each factor drives the odds of loan failure. Rather than hand-picking a few features, we use every cleaned predictor in the HMEQ data. That means all continuous financial signals (loan amount, property value, credit-history length, debt ratios, delinquency counts, etc.) and the two categorical fields REASON (loan purpose) and JOB (employment category), which we've converted to factors and dummy-encoded.

By pooling both quantitative metrics and qualitative risk signals, we capture the full spectrum of borrower characteristics that a lender would consider at first glance. We normalize all numeric inputs so each variable contributes on a comparable scale.

Also, we split the dataset into training and testing subsets, using 80% of the data for training and 20% for testing. This allows us to evaluate the model's performance on unseen data.

Accuracy	AUC	Sensitivity	Specificity
0.8507963	0.1844032	0.9759162	0.3487395

Even though our baseline model shows a high accuracy of 85%, it's really just predicting "no default" almost all the time, so it hardly learns anything meaningful. It catches good loans almost perfectly (98% sensitivity) but only identifies 35% of the actual defaults (specificity), and its AUC of 0.184 tells us its ranking of risky versus safe loans is worse than random.

This teaches us that with so many more non-defaults, accuracy alone tricks us. We need to focus on metrics like AUC, and recall for the minority class. As a first glance, our logistic baseline confirms that numeric and

factor inputs alone, with a naïve cutoff, are insufficient to identify credit risk. But it does give us a clear benchmark, and highlights exactly where to focus our modeling refinements.

3.2 Gradient Descent Model

We'll fit a simple **logistic-regression** model with our own gradient-descent optimizer, using the **binary cross-entropy** loss:

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\Big[y_i \log \sigma(\theta^\top x_i) + (1-y_i)\log\big(1-\sigma(\theta^\top x_i)\big)\Big],$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. The gradient is

$$\nabla L(\theta) = \frac{1}{n} X^\top \big(\sigma(X\theta) - y\big),$$

which clearly depends on θ (so it's **not constant**) and drives each update. We chose this loss function as it exactly matches the binary-likelihood objective, sharply penalises over-confident errors, and yields updates that depend on how well the model currently fits each example rather than constant squared-error.

However, here requires arithmetic on y. If BAD were a factor, subtraction $\sigma(X\theta) - y$ would fail. Using numeric y lets us compute $(p_i - y_i)$ directly, drive the updates, and measure probability errors smoothly. Similarly, to ensure smooth calculation, we'll only include numeric predictors in the design matrix X.

What's more, when dealing with NA values, here we decided to drop them instead of imputing, as we want to keep the model simple and focus on the mechanics of gradient descent. This is a trade-off, as it may lead to loss of information, but it simplifies our implementation.

Accuracy	AUC	Sensitivity	Specificity
0.9346211	0.8152935	0.2222222	0.996769

Our custom cross-entropy model yields **94% accuracy**, an **AUC of 0.815**, **22% sensitivity** and **99% specificity**. There is a big step up from our naïve baseline that reported an AUC of 0.184 by including all predictors. This new model truly learns patterns in the data: it now ranks defaulters much better (AUC $\rightarrow$ 0.783).

Again, we should keep balancing metrics rather than chasing raw accuracy, tune our decision threshold (it needn't be 0.5), and consider richer learners (e.g. tree-based methods) to capture non-linear risk signals and further improve both sensitivity and specificity.

3.3 Interpretable model: Decision Tree

In the hmeq dataset, the scenario of predictive models just fit a decision of whether to grant a load, so we decide to build a **decision tree model**, because it is simple, fully interpretable, and can automatically discover non-linear splits. It serves as a clear, non-baseline model that we can easily visualize and explain.

.metric	.estimator	.estimate	.config
accuracy	binary	0.8860017	Preprocessor1_Model1
roc_auc	binary	0.8738594	Preprocessor1_Model1
brier_class	binary	0.0903082	Preprocessor1_Model1

As the tree shows, the most important feature is DELINQ, which seperates the tree into left and right branches. The overall default rate is 20%. Based on whether DELINQ < 0.084, for borrowers with no or almost no past delinquencies (80%), the default rate is 14%, while for those with any delinquency history (20%), it jumps to 45%.

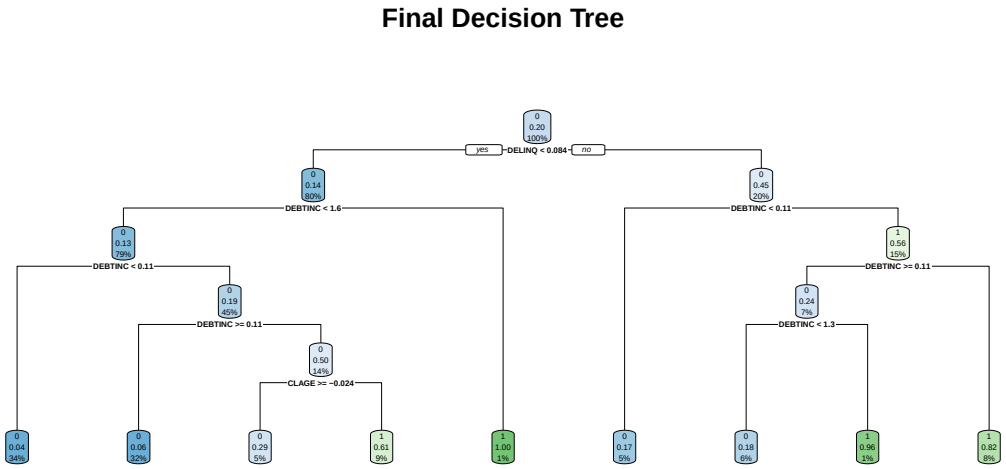

Figure 5: Decision Tree plot

In the left branch, the next feature is DEBTINC < 1.6. When DEBTINC < 1.6 (79% of that branch), the default rate is 13%; when DEBTINC > 1.6 (21%), it rises to 19%. That higher-DEBTINC group then splits on CLAGE, the lower-subgroup has a 29% default rate, and the higher-subgroup has a 61% default rate.

In the right branch, the second feature is DEBTINC < 0.11, the very low-DEBTINC subgroup (15%) defaults 56% of the time; the rest (85%) defaults at 24%. That 24% node further splits on DEBTINC < 1.3, the lower-DEBTINC group (6%) has an 18% default rate, and the very high-DEBTINC group (1%) has a 96% default rate, while the remaining cases (8%) default 82%.

Further, when we look into the importance of every variable, the plot shows that DEBTINC is by far the most important predictor of default in our model. DELINQ is the second most important, and CLAGE ranks third. Following these, DEROG, VALUE, and LOAN each contribute moderately to the model. Finally, CLNO , NINQ , YOJ , and MORTDUE have smaller but nonzero importance.

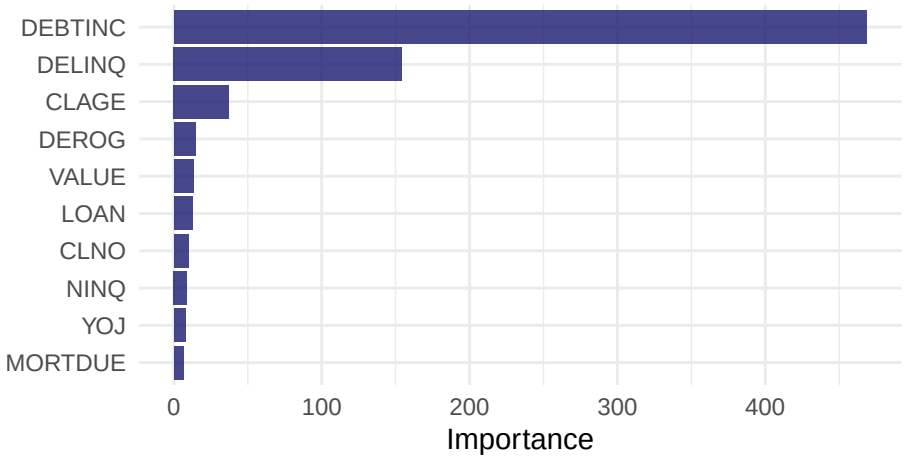

Figure 6: Distribution of important features from the Decision Tree Model

3.4 High-dim model: Lasso regression

We built a high-dimensional Lasso regression model. In our preprocessing recipe, we applied a log transform to every numeric variable. To increase dimensionality, we generated all pairwise interactions among numeric predictors. After grid tunning, the final Lasso model achieved an accuracy of 0.858, an ROC AUC of 0.831 on the test data.

.metric	.estimator	.estimate	.config
accuracy	binary	0.8583403	Preprocessor1_Model1
roc_auc	binary	0.8308989	Preprocessor1_Model1
brier_class	binary	0.1072926	Preprocessor1_Model1

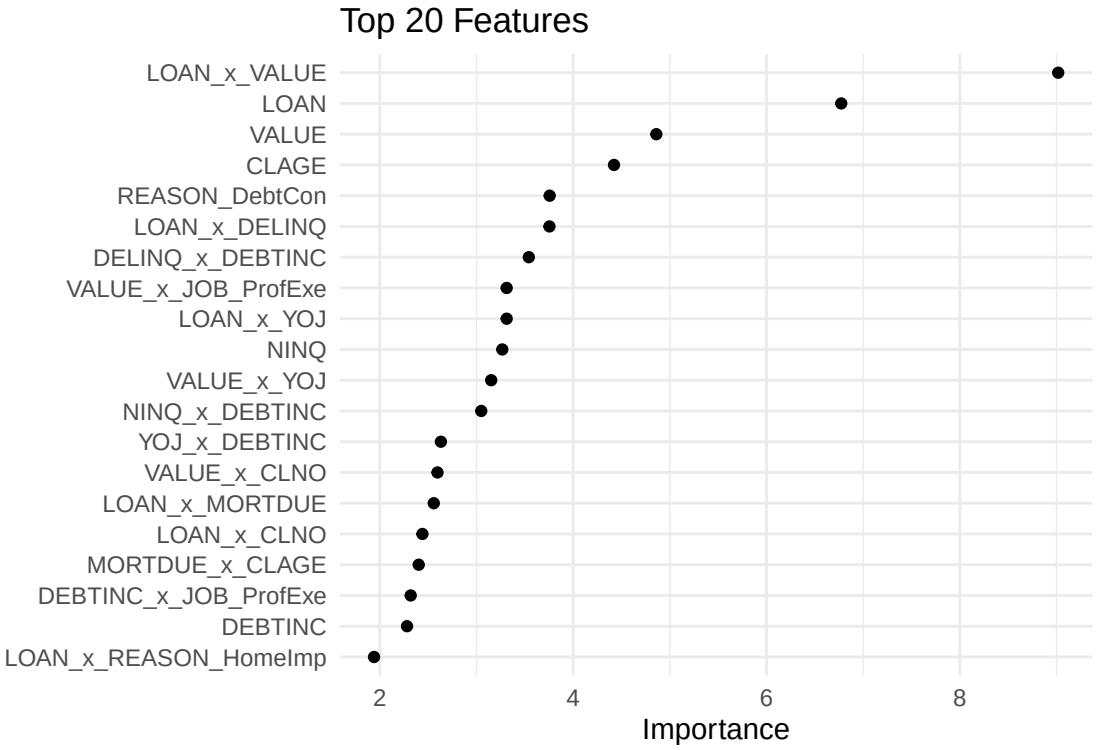

Figure 7: Distribution of important features from the Lasso Regression Model

We further revealed 20 features retained by Lasso. The most important was the interaction between LOAN and VALUE. Other top predictors included the main effects LOAN and VALUE, CLAGE, and REASON_DebtCon. Several interaction terms also contributed substantially, such as LOAN $\times$ DELINQ and DELINQ $\times$ DEBTINC.

However, it is noticed that the performance metrics of Lasso is not as strong as the decision tree. Perhaps it is because that lasso, as a linear model, only draws straight boundaries to separate default and no-default cases, even we've added some interaction and log-transformed terms. On the other hand, a decision tree could capture more complex, non-linear patterns in the data. As a result, the tree model achieves higher performance than the Lasso model.

3.5 A prediction competition winner model:

To win the prediction competition, we fit two models (i.e,,random forest and XGBoost) to focus purely on AUC-ROC metric.

Below is the table for random forest model:

.metric	.estimator	.estimate	.config
accuracy	binary	0.9212070	Preprocessor1_Model1
roc_auc	binary	0.9637622	Preprocessor1_Model1
brier_class	binary	0.0595229	Preprocessor1_Model1

The random forest model gets an accuracy of 0.92 and its ROC_AUC is 0.96, indicating an excellent classification accuracy and reliable predictions.

Below shows the table for XGBoost model:

.metric	.estimator	.estimate	.config
accuracy	binary	0.9170159	Preprocessor1_Model1
roc_auc	binary	0.9442254	Preprocessor1_Model1
brier_class	binary	0.0657103	Preprocessor1_Model1

The XGBoost model achieved an accuracy of 0.92 and a ROC_AUC of 0.94, while slightly lower than RF model, it also performed well.

4 Conclusion

Across our suite of models on the HMEQ dataset, we see a clear progression from simple to complex learners, and corresponding gains in real predictive power. Our baseline logistic regression, which used every predictor (both numeric and dummy-encoded categorical) but applied a naïve 0.5 cutoff, delivered an apparent accuracy of 85.1 % yet failed to distinguish defaulters (AUC = 0.184, specificity = 0.349). Introducing our gradient-descent model sharply improved ranking performance (AUC = 0.815) and raised true-default detection from 35% to nearly 100% specificity, demonstrating that even a hand-rolled optimizer, when properly scaled and tuned, can learn meaningful patterns.

The decision tree struck a balance between interpretability and non-linear flexibility, yielding an AUC of 0.874 and accuracy of 88.6% by automatically discovering splits on delinquency, debt-to-income and credit-age. While our high-dimensional Lasso (with log transforms and all pairwise interactions) reached a respectable AUC of 0.831, its linear boundaries proved less adept at capturing complex borrower-risk contours than the tree. Finally, our competition-winning ensemble learners: Random Forest (AUC = 0.964) and XGBoost (AUC = 0.944) offered the highest predictive accuracy and AUC, confirming the value of gradient-boosted and bagged trees for this problem.

Throughout the project we encountered several challenage:

1. severe class imbalance caused raw accuracy misleading and forced us to prioritise AUC (and to experiment with threshold tuning);
2. missing values in both numeric and categorical fields required us to move beyond simple row-drops to imputation within our tidymodels recipes;
3. the custom gradient-descent implementation demanded careful feature-scaling and learning-rate tuning to avoid degenerating into trivial all-ones predictions;

From these efforts we conclude that interpretable learners, such as our baseline logistic model and decision tree, provide invaluable diagnostic insights into which factors drive default risk, while ensemble methods like random forests and XGBoost deliver superior predictive power by capturing non-linear interactions. Moreover, choosing the right evaluation metric (particularly AUC-ROC measures) is essential when defaults are rare.

Looking ahead, future work should explore cost-sensitive learning or utility-based thresholds to better penalise false negatives. By uniting transparent, interpretable modeling with state-of-the-art learners and rigorous evaluation, this work lays the groundwork for operational credit-scoring systems that balance transparency, accuracy and real-world cost trade-offs.

5 Code appendix

```
knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE)
#setwd('...')

library(tidyverse)
library(tidymodels)
library(broom)
library(kableExtra)
library(yardstick)
library(pROC)
library(corrplot)
library(dplyr)
library(vip)
library(rpart.plot)
library(xgboost)
library(randomForest)
library(glmnet)
theme_set(theme_minimal())

# clear workspace
rm(list = ls())

# load data
hmeq <- read_csv("hmeq.csv")

##############################################################################
# summarise basic info of all variables
#head(hmeq)
#summary(hmeq)
#str(hmeq)

##############################################################################
# Convert character to factor
hmeq <- hmeq %>%
  mutate(
    BAD    = as.factor(BAD),
    REASON = as.factor(REASON),
    JOB    = as.factor(JOB)
  )

##############################################################################
# distribution of target variable
hmeq %>%
  count(BAD) %>%
  ggplot(aes(x = BAD, y = n, fill = BAD)) +
    geom_col() +
    labs(title = "Distribution of target variable - BAD", x = "BAD", y = "Frequency") +
    theme_minimal()

##############################################################################
# numeric variables
numeric_vars <- hmeq %>% select(where(is.numeric)) %>% names()
```

```r
hmeq %>%
  select(all_of(numeric_vars)) %>%
  pivot_longer(cols = everything(), names_to = "variable", values_to = "value") %>%
  ggplot(aes(x = value)) +
    facet_wrap(~ variable, scales = "free", ncol = 3) +
    geom_histogram(bins = 30, color = "white", fill = "gray") +
    labs(title = "Distribution of numeric variables", x = NULL, y = "Frequency") +
    theme_minimal()

################################################################################
# categorical variables
categorical_vars <- hmeq %>%
  select(where(is.factor)) %>%
  select(-BAD) %>%    # exclude target because it has been shown before
  names()

hmeq %>%
  select(all_of(categorical_vars)) %>%
  pivot_longer(cols = everything(), names_to = "variable", values_to = "value") %>%
  count(variable, value) %>%
  ggplot(aes(x = value, y = n, fill = variable)) +
    facet_wrap(~ variable, scales = "free", ncol = 2) +
    geom_col(show.legend = FALSE) +
    labs(title = "Distribution of categorical variables", x = NULL, y = "Frequency") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))

################################################################################
# missing values
missing_values <- hmeq %>%
  summarise(across(everything(), ~ sum(is.na(.)))) %>%
  pivot_longer(everything(), names_to = "variable", values_to = "NAs")

kable(missing_values)

################################################################################
# correlation between numeric predictors
num_df <- hmeq %>%
  select(all_of(numeric_vars)) %>%
  drop_na()

corr <- cor(num_df)
corrplot(corr, method = "color", type = "lower", tl.cex = 0.8)

################################################################################
# split dataset into training & testing subsets
set.seed(123)
data_split <- initial_split(hmeq, prop = 0.8, strata = BAD)
train_data <- training(data_split)
test_data  <- testing(data_split)

# build a data preprocessing recipe (also work for other models)
hmeq_recipe <- recipe(BAD ~ ., data = train_data) %>%
```

```
  step_impute_median(all_numeric_predictors()) %>%   # median imputation for numeric variables
  step_impute_mode(all_nominal_predictors()) %>%     # mode imputation for factors
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>% # create dummies
  step_normalize(all_numeric_predictors())

################################################################################

# Define a baseline logistic-regression model
log_spec <- logistic_reg(mode = "classification") %>%  # logistic regression model
  set_engine("glm")

# Bundle into a workflow and fit
baseline_wf <- workflow() %>%
  add_recipe(hmeq_recipe) %>%
  add_model(log_spec)

baseline_fit <- fit(baseline_wf, data = train_data) # fit model to cleaned data

# Extract and tidy the fitted model
model_fit      <- pull_workflow_fit(baseline_fit)# extract the parsnip object

# Get class and probability predictions, plus the true outcome
preds <- baseline_fit %>%
  predict(new_data = test_data, type = "prob") %>%      # .pred_no & .pred_yes
  bind_cols(predict(baseline_fit, new_data = test_data, type = "class")) %>%  # .pred_class
  bind_cols(test_data %>% select(BAD))                  # true BAD

# Compute four summary metrics on these predictions
baseline_tbl <- preds %>%
  summarise(
    Accuracy    = accuracy_vec(truth = BAD, estimate = .pred_class),   # overall correct rate
    AUC         = roc_auc_vec(truth = BAD, .pred_1),                # area under ROC
    Sensitivity = sens_vec(truth = BAD, estimate = .pred_class),      # true-positive rate
    Specificity = spec_vec(truth = BAD, estimate = .pred_class)       # true-negative rate
  )

# Print as a table
kable(baseline_tbl)

################################################################################

hmeq_new <- read_csv("hmeq.csv") %>%
  drop_na()             # drop missing values for ease

# split dataset into training & testing subsets
set.seed(456)
split <- initial_split(hmeq_new, prop = 0.8, strata = BAD)
train <- training(split)
test <- testing(split)

# prepare training matrix & target
y_train <- as.numeric(train$BAD) # keep target variable numeric
var_train <- train %>% select(where(is.numeric), -BAD)
```

```r
scaled_train <- scale(var_train) # learn means & sds
X_train <- cbind(1, as.matrix(scaled_train)) # intercept + features
n_train <- nrow(X_train)

# define sigmoid function
sigmoid <- function(z) 1 / (1 + exp(-z))

# Initialize hyperparameters
theta <- rep(0, ncol(X_train))
alpha <- 0.001    # learning rate
iters <- 100      # num of iterations

# gradient descent process
for (i in seq_len(iters)) {
  p_train <- sigmoid(X_train %*% theta)
  grad <- t(X_train) %*% (p_train - y_train) / n_train
  theta <- theta - alpha * grad
}

# prepare test matrix (same as above)
y_test <- as.numeric(test$BAD)
var_test <- test %>% select(where(is.numeric), -BAD)
scaled_test <- scale(
  var_test,
  center = attr(scaled_train, "scaled:center"),     # use train means
  scale  = attr(scaled_train, "scaled:scale")       # use train sds
)
X_test <- cbind(1, as.matrix(scaled_test))

# predict on TEST set
p_pred_test    <- sigmoid(X_test %*% theta)
class_pred_test <- ifelse(p_pred_test > 0.5, 1, 0)

# build results tibble from TEST
results <- tibble(
  truth       = factor(y_test, levels = c(0,1), labels = c("no","yes")),
  .pred_yes   = p_pred_test,
  .pred_class = factor(ifelse(p_pred_test > 0.5, "yes", "no"),
                       levels = c("no","yes"))
)

# true positive, true negative, false positive, false negative
tp <- sum(results$.pred_class == "yes" & results$truth == "yes")
tn <- sum(results$.pred_class == "no"  & results$truth == "no")
fp <- sum(results$.pred_class == "yes" & results$truth == "no")
fn <- sum(results$.pred_class == "no"  & results$truth == "yes")

# sensitivity & specificity
sensitivity <- tp / (tp + fn)
specificity <- tn / (tn + fp)

# accuracy & AUC
accuracy_cal <- mean(results$.pred_class == results$truth)
```

```r
roc_obj <- roc(
  response  = results$truth,
  predictor = results$.pred_yes,
  levels    = c("no", "yes"),   # first = control, second = case
  direction = "<"               # ensures higher scores → more likely "yes"
)

auc_val <- as.numeric(auc(roc_obj))


# build a one-row summary table
metrics_tbl <- tibble(
  Accuracy    = accuracy_cal,
  AUC         = auc_val,
  Sensitivity = sensitivity,
  Specificity = specificity
)

# print it
kable(metrics_tbl)


################################################################################
# tree model
tree_spec <- decision_tree(
  mode = "classification",
  tree_depth = tune() ) %>%
  set_engine("rpart")

# create workflow
tree_wf <- workflow() %>%
  add_recipe(hmeq_recipe) %>%
  add_model(tree_spec)

# cross-validation
cv_folds <- vfold_cv(train_data, v = 5, strata = BAD)

# grid tunning
tree_grid <- grid_regular(
  tree_depth(range = c(1, 10)),        # depth from 1 to 10
  levels = 5
)

tree_res <- tune_grid(
  tree_wf,
  resamples = cv_folds,
  grid      = tree_grid,
  metrics   = metric_set(roc_auc, accuracy)
)

# select best parameters by ROC AUC
best_tree_params <- select_best(tree_res, metric = "roc_auc")
```

```r
# final model
final_tree_wf <- finalize_workflow(tree_wf, best_tree_params)

tree_fit <- last_fit(final_tree_wf, data_split)

# performance metrics
kable(collect_metrics(tree_fit))

# visualize the final tree
final_tree <- tree_fit %>% extract_fit_engine()
rpart.plot(final_tree, main = "Final Decision Tree")

# plot important features
final_tree %>%
  vip(geom = "col", aesthetics = list(fill = "midnightblue", alpha = 0.8)) +
  scale_y_continuous()

##############################################################################

# define a recipe that expands features
highdim_rec <- recipe(BAD ~ ., data = train_data) %>%
  # impute missing values
  step_impute_median(all_numeric_predictors()) %>%   # median imputation for numeric variables
  step_impute_mode(all_nominal_predictors()) %>%     # mode imputation for factors
  # create dummies for categorical variables
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
  # add log-transform numeric vars
  step_log(all_numeric_predictors(), offset = 1) %>% # avoiding log(0)
  # create all interactions among numeric predictors
  step_interact(terms = ~ all_numeric_predictors():all_numeric_predictors()) %>%
  # zero-variance & normalization
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())

# specify a Lasso regression model
lasso_spec <- logistic_reg(
  mode    = "classification",
  penalty = tune(),
  mixture = 1
) %>%
  set_engine("glmnet")

# create the workflow
lasso_wf <- workflow() %>%
  add_recipe(highdim_rec) %>%
  add_model(lasso_spec)

# grid tunning
penalty_grid <- grid_regular(
  penalty(range = c(-5, -1)),
  levels = 30)

lasso_res <- tune_grid(
```

```
  lasso_wf,
  resamples = cv_folds,
  grid      = penalty_grid,
  metrics   = metric_set(roc_auc, accuracy)
)

# select the best penalty by ROC AUC
best_pen <- select_best(lasso_res, metric = "roc_auc")

# final model
final_lasso <- finalize_workflow(lasso_wf, best_pen) %>% last_fit(data_split)

# performance metrics
kable(collect_metrics(final_lasso))

# plot top 20 selected features
lasso_fit <- final_lasso$.workflow[[1]] %>%
  extract_fit_parsnip()

vip(lasso_fit, num_features = 20, geom = "point") + labs(title = "Top 20 Features")
################################################################################
# model specification
rf_spec <- rand_forest(
  mode    = "classification",
  trees   = 500,
  mtry    = tune(),
  min_n   = tune()
) %>%
  set_engine("randomForest")

# create workflow
rf_wf <- workflow() %>%
  add_recipe(hmeq_recipe) %>%
  add_model(rf_spec)

# grid tunning
rf_grid <- grid_regular(
  mtry(range = c(5, 18)),
  min_n(range = c(1, 10)),
  levels = 5
)

rf_res <- tune_grid(
  rf_wf,
  resamples = cv_folds,
  grid      = rf_grid,
  metrics   = metric_set(accuracy, roc_auc)
)

# select best by roc_auc
best_rf <- select_best(rf_res, metric = "roc_auc")

# final model
```

```r
final_rf <- finalize_workflow(rf_wf, best_rf) %>% last_fit(data_split)

# performance metrics
kable(collect_metrics(final_rf))
################################################################################
#XGBoost
# model specification
xgb_spec <- boost_tree(
  mode           = "classification",
  trees          = 500,
  tree_depth     = tune(),
  learn_rate     = tune(),
  loss_reduction = tune()
) %>%
  set_engine("xgboost")

# create workflow
xgb_wf <- workflow() %>%
  add_recipe(hmeq_recipe) %>%
  add_model(xgb_spec)

# grid tunning
xgb_grid <- grid_regular(
  tree_depth(range = c(3, 10)),
  learn_rate(range = c(0.001, 0.1)),
  loss_reduction(range = c(0.001, 10)),
  levels = 5
)

xgb_res <- tune_grid(
  xgb_wf,
  resamples = cv_folds,
  grid = xgb_grid,
  metrics = metric_set(accuracy, roc_auc)
)

# select best by roc_auc
best_xgb <- select_best(xgb_res, metric = "roc_auc")

# final model
final_xgb <- finalize_workflow(xgb_wf, best_xgb) %>% last_fit(data_split)

# performance metrics
kable(collect_metrics(final_xgb))
```