Group-26 AAD-Assignment-2

Omar, Eloise, Alina, Sue

2025-05-20

Contents

2.1 Nonlinear model vs non-parametric model

2.1.1 GAM vs KNN approach

GAM:

Pros

- **Flexibility to capture non-linearity**. In a GAM we replace each linear term $\beta_k x_k$ with a smooth function $f_k(x_k)$, meaning we can automatically model non-linear relationships between each predictor and the response without having to manually try out different transformation on each variable individually.
- **Non-linear fits may be more accurate**. The ability of creating this non-linear predictor and response relationship may make our model more accurate when predicting the medianHousingValue.
- **Interperability from additivity**. Since a GAM is additive, we can examine the effect of each X_j on Y individually while holding all of the other variables fixed. Therefore, we can understand each variables individual effect on house value.
- **Control over smoothness**. Each f_j comes with an associated smoothing parameter (or degrees of freedom), making it straightforward to trade bias and variance (e.g. via cross-validation).

Cons

- **Additivity assumption restriction**. If there are strong synergistic effects like between location (longitude/latitude) and income –an additive model will not capture them unless explicit interaction terms $X_j \times X_k$ are included or low-dimensional interactions function of the form $f_{jk}(X_j, X_k)$ are manually introduced.
- **Computational cost**. Fitting this model to over ~$20,600$ observations and multiple smoothers can be very slow when compared to fitting a single parametric method.

KNN:

Pros

- **Completely non-parametric**. KNN makes no assumptions about the form of $f(X)$, allowing the model to potentially fit better than a parametric model. At a point x_0 KNN averages the responses of the K closest training blocks: $\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i$, where N_0 is the set of the K nearest neighbours.
- **Control of bias-variance**. A small K yields a very flexible, low-bias but high-variance fit; large K yields a smoother, lower-variance fit.

Cons

- **Dimensoniality constraints**. As the number of predictors grows, the "nearest" neighbours tend to be far away in a high-dimensional space, so KNN's performance degrades rapidly as the number of predictors grow.
- **Distance metric sensitivity**. With a KNN you must scale the numeric features and encode the categorical features (e.g. oceanProximity) carefully. Otherwise poorly scaled or encoded features can dominate the distance of the nearest neighbours calculation.
- **Computationally intensive with predictions**. For each new group of predictions all ~$20,600$ must be computed to find the K nearest, which can be very intensive and slow.
- **Low interperability**. There is no simple way to explain a KNN prediction beyond pointing to the raw neighbours and their average.

In this housing-price context:

- **GAM:** it is likely to give an interpretable model with, as we can analyse the partial-effects (e.g. how median income or ocean proximity individually affects price), and we can capture smooth non-linear trends.

- **KNN:** can capture complex interactions, between predictors, automatically, but with eight predictors (including a categorical one) it may run into high-dimensionality issues, making distance-based averaging unstable and slow.

2.1.2 GAM vs KNN Regression Model

We will use the cleaned data (omitting all data points that contains "NA") instead of the original raw dataset. We split the data into a training set (80% of the original data) and a test set (20% of the original data).

```
# Load the data
data <- read.csv("Assignt2_data.csv")
data <- na.omit(data)
```

```
# Separate the training and test set out
set.seed(1)
train_index <- createDataPartition(data$medianHouseValue, p = 0.8, list = FALSE)
train_data <- data[train_index, ]
test_data  <- data[-train_index, ]
```

We use the same training and test set for this whole report.

Basic KNN Regression Model

The basic KNN regression model was trained using 5-fold cross-validation on the training dataset, with k tuned from 1 to 10. Preprocessing steps included centering and scaling to ensure features contribute equally to the distance metric.

```
# Basic KNN

knn_reg_model_1 <- train(
  medianHouseValue ~ .-id,
  data = train_data,
  method = "knn",
  preProcess = c("center", "scale"),
  tuneGrid = data.frame(k = 1:10),  # test k = 1 to 10
  trControl = trainControl(method = "cv", number = 5)
)

# Best k
print(knn_reg_model_1)
```

```
## k-Nearest Neighbors
##
## 16348 samples
##     9 predictor
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 13080, 13078, 13077, 13079, 13078
## Resampling results across tuning parameters:
```

```
##
##    k   RMSE      Rsquared   MAE
##     1  81963.63  0.5511857  54420.18
##     2  71586.59  0.6278105  48377.51
##     3  68378.57  0.6530505  46328.44
##     4  66666.50  0.6677736  45236.72
##     5  65783.58  0.6754535  44650.50
##     6  65241.29  0.6803072  44329.60
##     7  65008.23  0.6824027  44252.06
##     8  64925.97  0.6832080  44216.47
##     9  64762.67  0.6848104  44167.47
##    10  64546.74  0.6869904  44061.75
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 10.
```

```r
# KNN MSE calculation
knn_pred_1 <- predict(knn_reg_model_1, newdata = test_data)
knn_MSE_1 <- mean((knn_pred_1 - test_data$medianHouseValue)^2)
cat("The test MSE of basic KNN model is", knn_MSE_1, "\n")
```

```
## The test MSE of basic KNN model is 3983322872
```

```r
#3983322872
```

KNN Model Improvements

Addition and modification of features:

- Following the arguments in Assignment1, we've added the following features to the KNN regression model:
  - **bedroomsPerRoom**: captures housing density within units; higher ratios may indicate crowding, which could affect property value.
  - **incomePerRoom**: reflects economic status relative to housing size, providing a measure of affordability or wealth.
  - **distToLA** and **distToSF**: quantify proximity to major cities, which often correlates with higher housing prices.
  - **cosDirToLA/SF** and **sinDirToLA/SF**: encode directional information relative to LA and SF to help KNN detect spatial orientation patterns.
  - **cityProximityScore**: combines distance to both cities into a single metric, giving higher scores to locations near either city.
- We also encoded the categorical data to numeric form specifically for KNN so it can properly use it for distance calculations.
  - Converted the categorical variable `oceanProximity` to a factor to ensure proper handling of categories.
  - Applied one-hot encoding using `dummyVars()` to convert categorical variables into binary indicator variables.
  - Transformed the one-hot encoded matrices back into data frames for compatibility with modeling functions.

How we used the data:

- The KNN regression model was trained using the encoded and preprocessed training data (`train_data_encoded`), which includes both numeric and one-hot encoded categorical variables.
- We standardized features by centering and scaling (`preProcess = c("center", "scale")`) to ensure that distance calculations are not affected by different sizes of units.
- We selected the best `k` by performing cross validation (5-fold) over the range 1 to 15 (`tuneGrid = data.frame(k = 1:15)`).

```
# Import and clean data to get data_imp for model improvement
data_imp <- read.csv("Assignt2_data.csv")
data_imp <- na.omit(data_imp)

# bedroomsPerRoom
data_imp$bedroomsPerRoom <- data_imp$aveBedrooms / data_imp$aveRooms

# incomePerRoom
data_imp$incomePerRoom = data_imp$medianIncome / data_imp$aveRooms

# Compute distances
la_coords <- c(-118.24, 34.05)   # (longitude, latitude)
sf_coords <- c(-122.42, 37.77)

data_imp$distToLA <- apply(data_imp[, c("longitude", "latitude")], 1, function(coord) {
  distGeo(coord, la_coords) / 1000
})
data_imp$distToSF <- apply(data_imp[, c("longitude", "latitude")], 1, function(coord) {
  distGeo(coord, sf_coords) / 1000
})

# Compute directions
data_imp$dirToLA <- atan2(data_imp$latitude - la_coords[2], data_imp$longitude - la_coords[1])
data_imp$dirToSF <- atan2(data_imp$latitude - sf_coords[2], data_imp$longitude - sf_coords[1])

# Cosine and sine of direction
data_imp$cosDirToLA <- cos(data_imp$dirToLA)
data_imp$sinDirToLA <- sin(data_imp$dirToLA)
data_imp$cosDirToSF <- cos(data_imp$dirToSF)
data_imp$sinDirToSF <- sin(data_imp$dirToSF)
data_imp$dirToLA <- NULL
data_imp$dirToSF <- NULL

# Composite proximity score
data_imp$cityProximityScore <- 1 / (1 + data_imp$distToLA) + 1 / (1 + data_imp$distToSF)

# Add these to test and train
# Using the same training and test sets from before
train_data_imp <- data_imp[train_index, ]
test_data_imp  <- data_imp[-train_index, ]
```

```
# KNN with model improvements cont.
```

```r
# Hot encoding the variables

# Convert oceanProximity to factor
train_data_imp$oceanProximity <- as.factor(train_data_imp$oceanProximity)
test_data_imp$oceanProximity <- as.factor(test_data_imp$oceanProximity)

# One-hot encode using dummyVars
dummies <- dummyVars(~ ., data = train_data_imp)
train_data_encoded <- predict(dummies, newdata = train_data_imp)
test_data_encoded <- predict(dummies, newdata = test_data_imp)

# Convert to data frame
train_data_encoded <- as.data.frame(train_data_encoded)
test_data_encoded <- as.data.frame(test_data_encoded)

# KNN regression

set.seed(1)
knn_reg_model_2 <- train(
  medianHouseValue ~ .-id,
  data = train_data_encoded,
  method = "knn",
  preProcess = c("center", "scale"),
  tuneGrid = data.frame(k = 1:15),  # test k = 1 to 15
  trControl = trainControl(method = "cv", number = 5)
)

# Best k
print(knn_reg_model_2)
```

```
## k-Nearest Neighbors
##
## 16348 samples
##    22 predictor
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 13079, 13079, 13079, 13077, 13078
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    1  67229.74  0.6842298  43172.00
##    2  58960.44  0.7434965  38523.64
##    3  56746.61  0.7595084  37237.09
##    4  55717.26  0.7673094  36641.96
##    5  55099.02  0.7721771  36325.34
##    6  54659.74  0.7757243  36103.21
##    7  54361.30  0.7782753  36030.08
##    8  54432.13  0.7777877  36060.89
##    9  54362.43  0.7785184  36049.82
##   10  54411.06  0.7782571  36095.97
##   11  54567.02  0.7771765  36175.67
##   12  54768.53  0.7756700  36345.60
```

```
##   13  54960.45  0.7742252  36484.03
##   14  55048.35  0.7737057  36533.66
##   15  55165.68  0.7728772  36639.39
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

```
# KNN MSE calculation
knn_pred_2 <- predict(knn_reg_model_2, newdata = test_data_encoded)
knn_MSE_2 <- mean((knn_pred_2 - test_data_encoded$medianHouseValue)^2)
cat("The KNN MSE (improved) is", knn_MSE_2, "\n")
```

```
## The KNN MSE (improved) is 3060856986
```

```
# best now is: 3060856986
```

Basic GAM

- We used the same 80/20 train-test split as before to ensure consistency.
- The GAM model was trained using smoothing splines on original predictors to capture possible non-linear relationships.
- The select = TRUE argument enables automatic feature selection by penalising and shrinking irrelevant smooth terms toward zero. This is analogous to tuning the hyperparameter k in the Basic KNN regression, as both mechanisms helps to control model complexity and prevent overfitting.

```
# GAM

# train the GAM
set.seed(1)
gam_model_1 <- mgcv::gam(medianHouseValue ~ s(latitude) +
                           s(longitude) +
                    s(housingMedianAge) +
                      s(aveRooms) +
                    s(aveBedrooms) +
                      s(population) +
                    s(medianIncome) +
                      factor(oceanProximity),
                    data = train_data,
                    select = TRUE)

# Calculate MSE
gam_pred_1 <- predict(gam_model_1, newdata = test_data)
gam_MSE_1 <- mean((gam_pred_1 - test_data$medianHouseValue)^2)
cat("The Basic GAM MSE is", gam_MSE_1, "\n")
```

```
## The Basic GAM MSE is 4256244455
```

```
# 4257370057
# The Basic GAM MSE is 4256244455
```

Improved GAM

- Similar to the improved KNN Regression, the improved GAM model incorporates new features such as city proximity scores, distances, and directional terms to better capture spatial and socioeconomic effects.
- We also included a tensor product smooth, te(longitude, latitude), to flexibly capture complex spatial interactions between longitude and latitude.

```
# GAM improvements
train_data_imp = data_imp[train_index, ]
set.seed(1)
gam_model_2 <- mgcv::gam(medianHouseValue ~
                      s(housingMedianAge) +
                      s(aveRooms) +
                      s(aveBedrooms) +
                      s(population) +
                      s(medianIncome) +
                      te(longitude, latitude) +
                      s(cityProximityScore) +
                      s(bedroomsPerRoom) +
                      s(incomePerRoom) +
                      s(distToSF) +
                      s(distToLA) +
                      s(cosDirToLA) +
                      s(cosDirToSF) +
                      s(sinDirToLA) +
                      s(sinDirToSF),
                    data = train_data_imp,
                    select = TRUE)


# Calculate MSE
test_data_imp = data_imp[-train_index, ]
gam_pred_2 <- predict(gam_model_2, newdata = test_data_imp)
gam_MSE_2 <- mean((gam_pred_2 - test_data_imp$medianHouseValue)^2)
cat("The GAM MSE (improved) is", gam_MSE_2, "\n")
```

```
## The GAM MSE (improved) is 3386284239
```

```
# 3386501790
# The GAM MSE (improved) is 3386284239
```

2.1.3 Which model performs better?

Without adding any new features, KNN regression has a lower test MSE (3.98 billion) that is about 7% lower than the GAM (test MSE = 4.26). The baseline models were tuned - KNN with 5-fold cross-validation to select the optimal k, and GAM with automatic feature selection.

After introducing additional features and encoding the categorical variables, the KNN model's MSE dropped substantially by about 22% (to 3.06 billion). The MSE for GAM also reduced by 22% (to 3.34 billion). However, the KNN model still outperforms the GAM by a 8% lower test MSE.

KNN outperforms GAM both before and after the model improvements because its completely non-parametric nature that makes no assumptions about the functional form and averages responses of nearest neighbours (recall section 2.1.1 KNN Pros). This flexibility allows it to capture complex interactions introduced by new features such as cityProximityScore and the directional terms.

On the other hand, GAM is limited by its additive structure and smoothing assumptions, which restrict its ability to capture strong synergistic effects unless interactions are explicitly modelled (section 2.1.1 GAM Cons). Additionally, GAM's exponential family assumption (Gaussian here) on the response variable limits its flexibility to fully model complex, censored, and non-normal housing price data, making it less suitable compared to nonparametric models like KNN.

2.2 Classification models

2.2.1 New censoring boolean column

```
# Add censoring column and drop id and medianHouseValue
data_c <- data %>%
  na.omit() %>%
  mutate(censoring = ifelse(medianHouseValue == 500001, 1, 0)) %>%
  dplyr::select(-id, -medianHouseValue) %>%
  mutate(oceanProximity = as.factor(oceanProximity))



# This dataframe does the same changes as above but is with our model improvements
data_c_imp <- data_imp %>%
  na.omit() %>%
  mutate(censoring = ifelse(medianHouseValue == 500001, 1, 0)) %>%
  dplyr::select(-id, -medianHouseValue) %>%
  mutate(oceanProximity = as.factor(oceanProximity))
```

2.2.2 Two classification methods

Logistic Regression:

In this case logistic regression can model the log-odds of hitting the $\$500,001$ cap (censoring $= 1$) as a linear function of the predictors:

$$\log \frac{Pr(\text{censor} = 1|X_1, ..., X_p)}{1 - Pr(\text{censor} = 1|X_1, ..., X_p)} = \beta_0 + \beta_1 \text{ medianIncome} + \beta_2 \text{ housingMedianAge} + ...$$

This method is reasonable because:

1. **Ideal for binary classification problems**: Designed specifically for binary response variables, logistic regression is suitable since in this case censoring take values of either 1 (censored) or 0 (not censored).
2. **No strong distributional assumptions on $\boldsymbol{X}$**: Some of our variables may be skewed or heteroscedastic; logistic regression simply cares that the logit is linear in $\boldsymbol{X}$ (where $\boldsymbol{X} = (X_1, ..., X_p)$).
3. **Interpretable effects**: e.g. e^{β_1} tells us how one unit ($\$1,000$) increases in medianIncome multiplies the odds of a house price being censored.

4. **Flexibility**: It estimates the probability that a given input belongs to a particular class, which also provides us with the flexibility of choosing the threshold for which optimises the testing correct rate with the assistance of the ROC curve.

KNN-CV:

In this case KNN-CV is a non-parametric approach that classifies each block group by majority voting among its k closest neighbours in the predictor space. Formally, for a test point x_0, let N_0 be the indices of the k nearest training observations (by Euclidean distance):

$$Pr(\text{censor} = 1|x_0) = \frac{1}{k} \sum_{i \in N_0} I(\text{censor}_i = 1)$$

and we predict the class with the larger estimated probability. We choose k via cross-validation to pick the optimal bias-variance trade-off. \

This method is reasonable because:

1. **Captures nonlinearity**: If censoring depends on complex interactions and requires a non-linear decision boundary, KNN can adapt without specifying a specific model form.
2. **No parametric assumptions**: No distributional assumptions required–so it's robust to skewed, heteroscedastic variables.
3. **Tunable bias-variance trade-off via cross-validation**: The ability to choose k allows us to control the smoothness.
    - Small $k \implies$ very flexible (low bias, high variance)
    - Large $k \implies$ smoother (high bias, low variance)
    - Using cross-validation empirically estimates the error for each k, letting us pick a k value the minimises both over and under fitting

2.2.3 Which classification method performs better?

How we have used the given data:

- We used a random 80/20 training and test split of the dataset to produce the test error rates.
- We dropped the *id* and *medianHouseValue* columns to ensure we only use legitimate predictors when training and testing our model.
- The logistic model thresholds at 0.5, while KNN makes decisions based on the majority class among the nearest neighbours in the feature space.

```
# Logistic Regression without Model Improvements

train_data_c <- data_c[train_index, ]
test_data_c <- data_c[-train_index, ]

logit_fit_1 <- glm(censoring ~ longitude +
                   latitude +
                   medianIncome +
                   housingMedianAge +
                   oceanProximity +
```

```
                  aveRooms +
                  aveBedrooms,
                data = train_data_c,
                family = binomial)

logit_pred_1 <- as.numeric(predict(logit_fit_1,
                     newdata = test_data_c,
                     type = "response") > 0.5) # use 0.5 for now

error_rate <- mean(logit_pred_1 != test_data_c$censoring)

print(paste("Logistic Regression Prediction Error Rate:", round(error_rate, 3)))
```

```
## [1] "Logistic Regression Prediction Error Rate: 0.031"
```

```
# Confusion matrix
confusion_matrix <- confusionMatrix(as.factor(logit_pred_1), as.factor(test_data_c$censoring))
print(confusion_matrix)
```

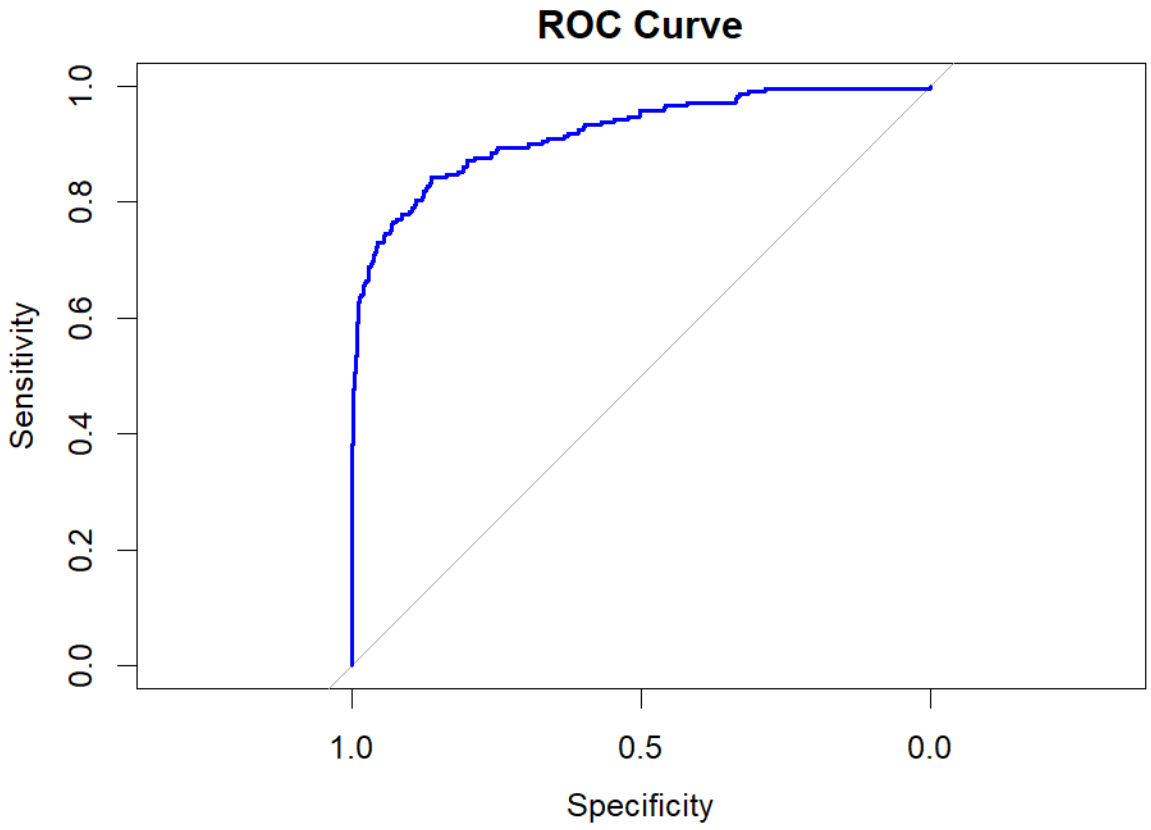
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3857  107
##          1   20  101
##
##                Accuracy : 0.9689
##                  95% CI : (0.9631, 0.974)
##     No Information Rate : 0.9491
##     P-Value [Acc > NIR] : 4.094e-10
##
##                   Kappa : 0.599
##
##  Mcnemar's Test P-Value : 2.325e-14
##
##             Sensitivity : 0.9948
##             Specificity : 0.4856
##          Pos Pred Value : 0.9730
##          Neg Pred Value : 0.8347
##              Prevalence : 0.9491
##          Detection Rate : 0.9442
##    Detection Prevalence : 0.9704
##       Balanced Accuracy : 0.7402
##
##        'Positive' Class : 0
##
```

```
# ROC
logit_pred_probs_1 <- predict(logit_fit_1, newdata = test_data_c, type = "response")
roc_curve_1 <- roc(test_data_c$censoring, logit_pred_probs_1)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_curve_1, main = "ROC Curve", col = "blue", lwd = 2)
```




```
# error rate: 0.031
```

```
# Logistic Regression with Model Improvements

train_data_c_imp <- data_c_imp[train_index, ]
test_data_c_imp <- data_c_imp[-train_index, ]

logit_fit_2 <- glm(censoring ~ longitude +
                   latitude +
                   medianIncome +
                   housingMedianAge +
                   oceanProximity +
                   aveRooms +
                   aveBedrooms +
                   cityProximityScore +
                   bedroomsPerRoom +
                   incomePerRoom +
                   distToSF +
                   distToLA +
                   cosDirToLA +
                   cosDirToSF +
```

```
                  sinDirToLA +
                  sinDirToSF,
                data = train_data_c_imp,
                family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
logit_pred_2 <- as.numeric(predict(logit_fit_2,
                     newdata = test_data_c_imp,
                     type = "response") > 0.5) # use 0.5 for now

error_rate <- mean(logit_pred_2 != test_data_c_imp$censoring)

print(paste("Logistic Regression Prediction Error Rate:", round(error_rate, 3)))
```

```
## [1] "Logistic Regression Prediction Error Rate: 0.029"
```

```
# Confusion matrix
confusion_matrix <- confusionMatrix(as.factor(logit_pred_2), as.factor(test_data_c_imp$censoring))
print(confusion_matrix)
```

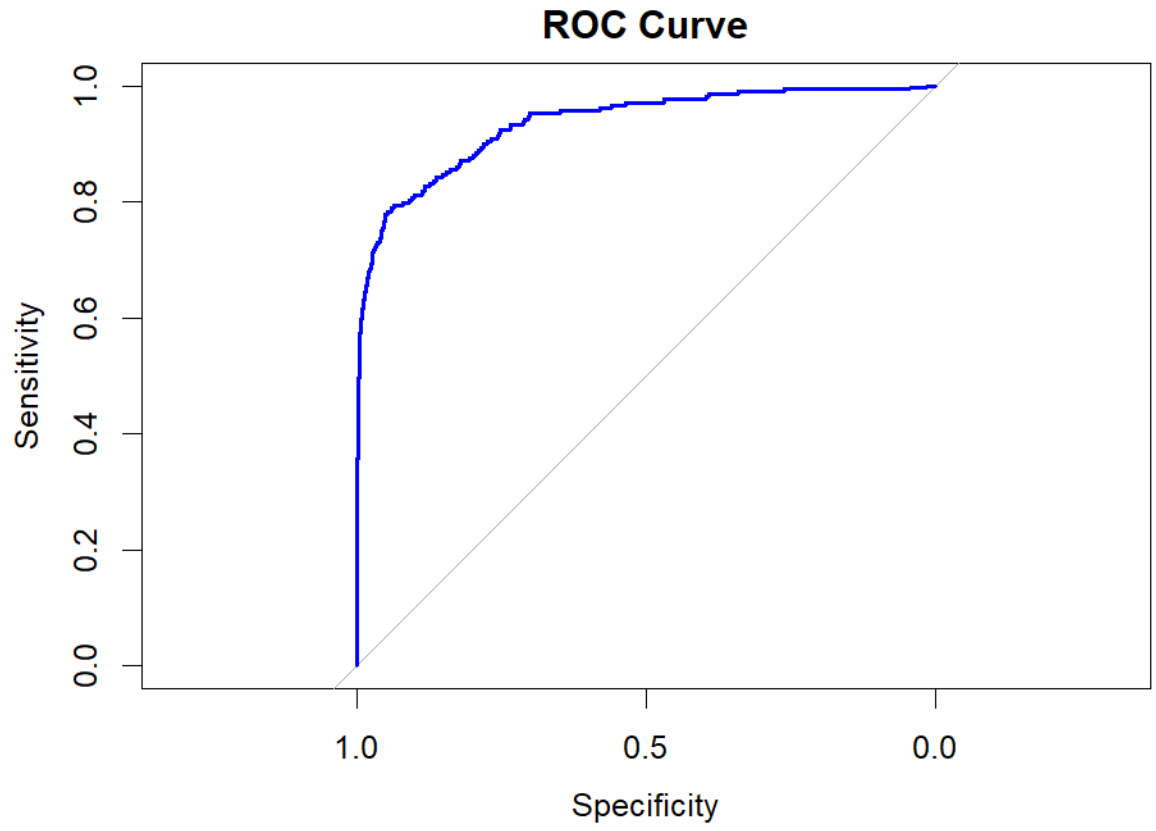
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3854   95
##          1   23  113
##
##                Accuracy : 0.9711
##                  95% CI : (0.9655, 0.976)
##     No Information Rate : 0.9491
##     P-Value [Acc > NIR] : 2.686e-12
##
##                   Kappa : 0.6426
##
##  Mcnemar's Test P-Value : 6.315e-11
##
##             Sensitivity : 0.9941
##             Specificity : 0.5433
##          Pos Pred Value : 0.9759
##          Neg Pred Value : 0.8309
##              Prevalence : 0.9491
##          Detection Rate : 0.9435
##    Detection Prevalence : 0.9667
##       Balanced Accuracy : 0.7687
##
##        'Positive' Class : 0
##
```

```
# ROC
logit_pred_probs_2 <- predict(logit_fit_2, newdata = test_data_c_imp, type = "response")
roc_curve_2 <- roc(test_data_c_imp$censoring, logit_pred_probs_2)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
plot(roc_curve_2, main = "ROC Curve", col = "blue", lwd = 2)
```




```
# error rate: 0.029
```

```
## KNN-CV Without Model Improvements

# Prepare the predictors (X) and response (Y)
train.X <- model.matrix(~ longitude +
                          latitude +
                          housingMedianAge +
                          aveRooms +
                          aveBedrooms +
                          population +
                          medianIncome +
                          oceanProximity,
                        data = train_data_c)[, -1]

test.X <- model.matrix(~ longitude +
                         latitude +
                         housingMedianAge +
                         aveRooms +
```

```
                          aveBedrooms +
                          population +
                          medianIncome +
                          oceanProximity,
                        data = test_data_c)[, -1]

train.Y <- train_data_c$censoring
test.Y <- test_data_c$censoring

# Use cross-validation on the training set to choose the best k
knn_rp <- rep(0, 15)
for (k in 1:15) {
  set.seed(5)
  knn.pred.cv <- knn.cv(train = train.X, cl = train.Y, k = k)
  knn_rp[k] <- mean(knn.pred.cv == train.Y)
}

k.cv <- which.max(knn_rp)
cat("Best k selected by CV is:", k.cv, "\n")
```

```
## Best k selected by CV is: 5
```

```
# Fit and predict on the test set using the best k
set.seed(5)
knn_pred_c_1 <- knn(train = train.X, test = test.X, cl = train.Y, k = k.cv)

# Compute confusion matrix and error rate
confusionMatrix(as.factor(knn_pred_c_1), as.factor(test.Y))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3876  199
##          1    1    9
##
##                Accuracy : 0.951
##                  95% CI : (0.944, 0.9575)
##     No Information Rate : 0.9491
##     P-Value [Acc > NIR] : 0.2994
##
##                   Kappa : 0.0783
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99974
##             Specificity : 0.04327
##          Pos Pred Value : 0.95117
##          Neg Pred Value : 0.90000
##              Prevalence : 0.94908
##          Detection Rate : 0.94884
##    Detection Prevalence : 0.99755
##       Balanced Accuracy : 0.52151
```

```
##
##        'Positive' Class : 0
##
```

```
(knn_err_c_1 = mean(knn_pred_c_1 != test.Y))
```

```
## [1] 0.04895961
```

```
cat("The KNN error rate is", round(knn_err_c_1, 4), "\n")
```

```
## The KNN error rate is 0.049
```

```
#The KNN error rate is 0.049
```

```
## KNN-CV with Model Improvements

# Prepare the predictors (X) and response (Y)
train.X <- model.matrix(~ longitude +
                          latitude +
                          medianIncome +
                          housingMedianAge +
                          oceanProximity +
                          aveRooms +
                          aveBedrooms +
                          cityProximityScore +
                          bedroomsPerRoom +
                          incomePerRoom +
                          distToSF +
                          distToLA +
                          cosDirToLA +
                          cosDirToSF +
                          sinDirToLA +
                          sinDirToSF,
                        data = train_data_c_imp)[, -1]

test.X <- model.matrix(~  longitude +
                          latitude +
                          medianIncome +
                          housingMedianAge +
                          oceanProximity +
                          aveRooms +
                          aveBedrooms +
                          cityProximityScore +
                          bedroomsPerRoom +
                          incomePerRoom +
                          distToSF +
                          distToLA +
                          cosDirToLA +
                          cosDirToSF +
                          sinDirToLA +
                          sinDirToSF,
                       data = test_data_c_imp)[, -1]
```

```r
train.Y <- train_data_c_imp$censoring
test.Y <- test_data_c_imp$censoring

# Use cross-validation on the training set to choose the best k
knn_rp <- rep(0, 15)
for (k in 1:15) {
  set.seed(5)
  knn.pred.cv <- knn.cv(train = train.X, cl = train.Y, k = k)
  knn_rp[k] <- mean(knn.pred.cv == train.Y)
}

k.cv <- which.max(knn_rp)
cat("Best k selected by CV is:", k.cv, "\n")
```

```
## Best k selected by CV is: 5
```

```r
# Fit and predict on the test set using the best k
set.seed(5)
knn_pred_c_2 <- knn(train = train.X, test = test.X, cl = train.Y, k = k.cv)

# Compute confusion matrix and error rate
confusionMatrix(as.factor(knn_pred_c_2), as.factor(test.Y))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3859  107
##          1   18  101
##
##                Accuracy : 0.9694
##                  95% CI : (0.9636, 0.9745)
##     No Information Rate : 0.9491
##     P-Value [Acc > NIR] : 1.419e-10
##
##                   Kappa : 0.603
##
##  Mcnemar's Test P-Value : 3.519e-15
##
##             Sensitivity : 0.9954
##             Specificity : 0.4856
##          Pos Pred Value : 0.9730
##          Neg Pred Value : 0.8487
##              Prevalence : 0.9491
##          Detection Rate : 0.9447
##    Detection Prevalence : 0.9709
##       Balanced Accuracy : 0.7405
##
##        'Positive' Class : 0
##
```

```r
(knn_err_c_2 = mean(knn_pred_c_2 != test.Y))
```

```
## [1] 0.03059976
```

```r
cat("The KNN error rate is", round(knn_err_c_2, 4), "\n")
```

```
## The KNN error rate is 0.0306
```

```r
#The KNN error rate is 0.0306
```

Note: We did not generate ROC curves for KNN because it does not provide continuous probability scores but rather hard class labels. ROC analysis requires varying the classification threshold across a range of values, which is not possible with the standard KNN output. While probabilistic adaptations of KNN exist, they were not necessary here since our objective was classification accuracy, not probability estimation.

2.2.4 Justifying the better classification method

Logistic regression outperformed KNN-CV in our 80/20 hold-out test, with test errors of 0.0290 and 0.0306 respectively. Logistic regression is a parametric model, it assumes a specific relationship between predictors and the target, resulting in higher bias but lower variance. This rigidity works in its favour when the underlying structure of the data is close to linear, as it avoids fitting noise and reduces the risk of overfitting. KNN, on the other hand, is non-parametric and makes very few assumptions about the data's form. It is flexible and can capture complex, non-linear patterns, but that flexibility comes at the cost of high variance, especially in high-dimensional settings.

While KNN's flexibility can be useful, it becomes increasingly unstable in high-dimensional spaces where data points are more sparsely distributed and neighbourhoods lose their meaning. Specifically, the distance between the nearest and farthest neighbours shrinks, making it hard to define what counts as "close," which undermines the core logic of KNN. Even with cross-validation to choose the optimal k, KNN was still overfitting, likely due to this dimensionality issue. Logistic regression, despite its linear constraint, handled the complexity of the data more gracefully by not reacting too much to the noise.

The lower test error achieved by logistic regression shows that its higher bias was actually beneficial here, it provided a smoother, more stable model that generalised well. So while KNN theoretically offers more flexibility, in this case it became a liability, and logistic regression ultimately struck the better balance. To note, the test error is only marginally larger for the KNN, and both are relatively low so it is clear that both models are effective, just for this random seed, logistic regression is the winner.

2.3 A hybrid approach

2.3.1 Dicussing feasibility of the approach

Estimate using improved KNN and then apply the censoring from logistic. More explanation necessary

To address the censoring issue in the dataset, a hybrid approach that combines regression and classification models is both feasible and potentially very effective. The best regression model from 2.1.3 is used to predict house values, while the best classifier from 2.2.2 identifies whether a given observation is censored; i.e. whether the true house value exceeds the $500,001 threshold. This setup allows us to adjust regression predictions when the classifier indicates censoring, helping to reduce the bias that occurs when censored values are treated as if they were exact rather than right censored.

The procedure involves four main steps. First, we use the KNN regression model to predict house prices. Second, a logistic regression classifier estimates the probability of censoring. Third, we adjust the regression predictions by setting them to 500001 for observations predicted to be censored. Finally, we evaluate the performance of this adjusted model using test MSE. This combined method leverages the strengths of both models and is a practical way to improve prediction accuracy in the presence of censoring.

2.3.2 test MSE of medianHousingValue using the approach

```
# Step 1: Predict house prices with KNN regression
knn_pred_2 <- predict(knn_reg_model_2, newdata = test_data_encoded)

# Step 2: Predict censoring with logistic regression
logit_pred_2 <- as.numeric(predict(logit_fit_2, newdata = test_data_c_imp, type = "response") > 0.5)

# Step 3: Adjust predicted house prices
# If censored predicted, clip to 500001
adjusted_preds <- ifelse(logit_pred_2 == 1, 500001, knn_pred_2)

# Step 4: Compute MSE using all observations
final_mse <- mean((adjusted_preds - test_data_encoded$medianHouseValue)^2)

cat("Final Test MSE:", round(final_mse, 2), "\n")
```

```
## Final Test MSE: 3077760238
```

```
# Final Test MSE: 3077760238
```

2.3.3 Comparison of the accuracy of this procedure to model in 2.1.3

The hybrid model resulted in a slightly higher test MSE (3.07 billion) compared to the original KNN regression model from 2.1.3 (3.06 billion). These results were obtained using a random 80/20 train-test split of the data the same split used throughout the course of this report. Given how close the two MSE values are, it's possible that on a different random sample, the hybrid model could perform slightly better. Overall, the difference is minimal, suggesting that both models offer comparable predictive performance in this context.

The reason the hybrid model does not consistently outperform KNN likely stems from how it handles censoring using a hard classification decision. Specifically, it clips predicted values to 500001 whenever the logistic classifier predicts an observation is censored. This is because logistic regression applies a strict decision boundary (at 0.5) to make this call, which introduces a sharp threshold with no gradual transition. This can be problematic near the boundary, where small changes in input may flip the classification, yet the model treats the output as definitively censored or not. If the classifier incorrectly predicts censoring, the hard cap forces the prediction to 500001, which can be significantly off from the true (uncensored) value. In contrast, the KNN model always outputs a continuous estimate. When the classifier misclassifies an uncensored point as censored, the true value is often still closer to the KNN prediction than to the fixed 500001. As a result, KNN may show better MSE performance, despite not explicitly addressing censoring.