Assignment 2 Walkthrough

Warren Mansur

Reminder: Purpose of Walkthrough

- The goal of this walkthrough is to help you successfully understand and complete Assignment 2.

Reminder: Purpose of Walkthrough

- The goal of this walkthrough is to help you successfully understand and complete Assignment 2.
- I briefly recap important concepts relevant to the assignment, then show you relevant code examples.

Reminder: Purpose of Walkthrough

- The goal of this walkthrough is to help you successfully understand and complete Assignment 2.
- I briefly recap important concepts relevant to the assignment, then show you relevant code examples.
- These code examples are not shown elsewhere (i.e. not in the instructor's live class, shared example code, or online modules).

Reminder: Live Classrooms

- It's important to attend Prof. Joner's live classrooms to learn the core concepts. The sessions are thorough and deep.

Reminder: Live Classrooms

- It's important to attend Prof. Joner's live classrooms to learn the core concepts. The sessions are thorough and deep.
- This walkthrough recaps some concepts to focus on the practical application for the assignment, but does not go into the same conceptual depth.

Reminder: Live Classrooms

- It's important to attend Prof. Joner's live classrooms to learn the core concepts. The sessions are thorough and deep.
- This walkthrough recaps some concepts to focus on the practical application for the assignment, but does not go into the same conceptual depth.
- Prof. Joner provides one sample R file per week to get you started. They demonstrate use of some important libraries and functions, but still require you to apply the code to the specific assignment scenarios.

Reminder: Submitting

- It's important to provide two files – one for your answers without the code, and the other with the code.

Reminder: Submitting

- It's important to provide two files – one for your answers without the code, and the other with the code.
- This keeps your work and answers separate, helping ensure that you receive credit where it is due.

Reminder: Submitting

- It's important to provide two files – one for your answers without the code, and the other with the code.
- This keeps your work and answers separate, helping ensure that you receive credit where it is due.
- There have been occasions in past runnings where the work over-cluttered the answers, and a few lost some points because of misidentification.

Reminder: Submitting

- It's important to provide two files – one for your answers without the code, and the other with the code.
- This keeps your work and answers separate, helping ensure that you receive credit where it is due.
- There have been occasions in past runnings where the work over-cluttered the answers, and a few lost some points because of misidentification.
- Use this YAML to hide code globally:

```yaml
# ----- in the document YAML -----
execute:
  echo:    false   # hide code globally
  warning: false   # hide warnings globally
```

Objects with Only Nominal Attributes

Table 1: Is Car C1 closer to C6 or C9?

ID	Make	Body Style	Fuel	Transmission	Color	Ownership
C1	Toyota	sedan	gas	automatic	white	first-owner
C2	Ford	truck	diesel	automatic	blue	corporate
C3	Honda	hatchback	gas	manual	red	second-owner
C4	BMW	coupe	gas	automatic	black	first-owner
C5	Tesla	sedan	electric	automatic	silver	corporate
C6	Hyundai	SUV	gas	automatic	white	rental
C7	Chevrolet	truck	gas	manual	red	second-owner
C8	Toyota	SUV	hybrid	automatic	blue	first-owner
C9	Ford	coupe	gas	manual	black	corporate

Hamming Distance Between Objects

- **Objective** – Quantify how similar two records are when every attribute is **nominal**.

Hamming Distance Between Objects

- **Objective** – Quantify how similar two records are when every attribute is **nominal**.
- **Representation** – Treat as n-element string of labels (only exact matches matter).

Hamming Distance Between Objects

- **Objective** – Quantify how similar two records are when every attribute is **nominal**.
- **Representation** – Treat as n-element string of labels (only exact matches matter).
- **Scaled Hamming Distance** – distance $=$ $\#$ mismatched attributes/$\#$ attributes

Hamming Distance Between Objects

- **Objective** – Quantify how similar two records are when every attribute is **nominal**.
- **Representation** – Treat as n-element string of labels (only exact matches matter).
- **Scaled Hamming Distance** – distance $=$ # mismatched attributes/# attributes
- $0 =$ all attributes match (identical profiles)

Hamming Distance Between Objects

- **Objective** – Quantify how similar two records are when every attribute is **nominal**.
- **Representation** – Treat as n-element string of labels (only exact matches matter).
- **Scaled Hamming Distance** – distance $=$ # mismatched attributes/# attributes
- $0 =$ all attributes match (identical profiles)
- $1 =$ all attributes differ (maximally dissimilar)

Distance Between Cars Code

```r
# Load the car-ownership data and compute scaled Hamming distances
cars <- data.frame(
  ID           = c("C1","C2","C3","C4","C5","C6","C7","C8","C9"),
  Make         = c("Toyota","Ford","Honda","BMW","Tesla","Hyundai","Chevrolet","Toyota","Ford"),
  Body_Style   = c("sedan","truck","hatchback","coupe","sedan","SUV","truck","SUV","coupe"),
  Fuel         = c("gas","diesel","gas","gas","electric","gas","gas","hybrid","gas"),
  Transmission = c("automatic","automatic","manual","automatic","automatic","automatic","manual","automatic","manual"),
  Color        = c("white","blue","red","black","silver","white","red","blue","black"),
  Ownership    = c("first-owner","corporate","second-owner","first-owner","corporate","rental","second-owner","first-owner","corporate"),
  stringsAsFactors = FALSE        # keep as plain strings for direct comparison
)

nominal_cols <- c("Make","Body_Style","Fuel","Transmission","Color","Ownership") # Columns to compare (all six are nominal)
hamming <- function(id_a, id_b) { # Scaled Hamming distance = proportion of mismatches across the six features
  i <- match(id_a, cars$ID)        # row number of id_a
  j <- match(id_b, cars$ID)        # row number of id_b
  mean(cars[i, nominal_cols] != cars[j, nominal_cols])
}

dist_C1_C6 <- hamming("C1", "C6") # Compare C1 to C6 and C9
dist_C1_C9 <- hamming("C1", "C9")
closer_car <- ifelse(dist_C1_C6 < dist_C1_C9, "C6", "C9") # Identify which car is closer to C1

list(distance_C1_C6 = dist_C1_C6, # Return a tidy summary
     distance_C1_C9 = dist_C1_C9,
     closer_to_C1   = closer_car)
```

Distance Between Cars Results

```
$distance_C1_C6
[1] 0.5

$distance_C1_C9
[1] 0.8333333

$closer_to_C1
[1] "C6"
```

Objects with Only Numeric Attributes

Table 2: Calculate Manhattan and Euclidean Distance for O1, O2

Object	Z1	Z2	Z3	Z4	Z5
O1	95	40	35	5	50
O2	82	70	16	4	49

Euclidean and Manhattan Distances Between Objects

- **Objective** – measure numeric dissimilarity between O1 and O2 across multiple attributes.

Euclidean and Manhattan Distances Between Objects

- **Objective** – measure numeric dissimilarity between O1 and O2 across multiple attributes.
- **Coordinate view** – treat each object as the point and each gap is computed component-wise.

Euclidean and Manhattan Distances Between Objects

- **Objective** – measure numeric dissimilarity between O1 and O2 across multiple attributes.
- **Coordinate view** – treat each object as the point and each gap is computed component-wise.
- **Manhattan distance** – sum of absolute gaps. Captures the cost of moving only along axes; less sensitive to one extreme attribute.

Euclidean and Manhattan Distances Between Objects

- **Objective** – measure numeric dissimilarity between O1 and O2 across multiple attributes.
- **Coordinate view** – treat each object as the point and each gap is computed component-wise.
- **Manhattan distance** – sum of absolute gaps. Captures the cost of moving only along axes; less sensitive to one extreme attribute.
- **Euclidean (straight-line) distance** – square-root of squared gaps. Squaring emphasizes large differences.

Distance Between O1, O2 Code

```r
# Load the two-row Z-values data set and compute Euclidean & Manhattan distances
z <- data.frame(
  Object = c("O1", "O2"),
  Z1 = c(95, 82),
  Z2 = c(40, 70),
  Z3 = c(35, 16),
  Z4 = c(5,  4),
  Z5 = c(50, 49)
)

# Keep only the numeric attributes for distance calculations
z_vals <- z[, -1]         # drop the identifier column

# Euclidean distance – dist() defaults to Euclidean for numeric data frames
euclidean_O1_O2 <- as.numeric(dist(z_vals)[1])

# Manhattan ( ℓ1 ) distance – specify method = "manhattan"
manhattan_O1_O2 <- as.numeric(dist(z_vals, method = "manhattan")[1])

# Return both distances in a tidy structure
list(euclidean = euclidean_O1_O2,
     manhattan = manhattan_O1_O2)
```

Distance Between O1, O2 Results

```
$euclidean

[1] 37.84178


$manhattan

[1] 64
```

Package Delivery Dataset

Table 3: Package Delivery Dataset – 8 Row Sample

ID	Dist	WtLb	Fragil	TwoDay	PkRsh	WthrAd	DrvExp	HndOff	UrbDst	Status
P1	939.5	106.2	0	0	1	0	4.2	0	1	LateMinor
P2	2377.0	23.3	0	0	1	0	2.6	0	1	LateMinor
P3	1831.3	86.7	0	0	0	0	27.3	0	0	OnTime
P4	1498.7	91.2	0	0	1	0	6.1	1	1	OnTime
P5	394.3	63.9	0	0	1	0	24.7	3	1	OnTime
P6	394.2	110.6	0	0	1	0	16.6	1	0	OnTime
P7	149.9	140.2	0	0	1	0	2.8	2	0	LateMinor
P19	1082.7	8.7	0	0	1	0	26.7	2	1	LateMajor

Building and Evaluating a k-Nearest Neighbor Classifier

Use the dataset **package_delivery.csv** (824 observations, 10 features; class = DELIV_STATUS).

1. Generate training and hold-out partitions, keeping $\frac{1}{3}$ of the data for hold-out.

Building and Evaluating a k-Nearest Neighbor Classifier

Use the dataset **package_delivery.csv** (824 observations, 10 features; class = DELIV_STATUS).

1. Generate training and hold-out partitions, keeping $\frac{1}{3}$ of the data for hold-out.
2. Fit a k-Nearest Neighbor model.

Building and Evaluating a k-Nearest Neighbor Classifier

Use the dataset **package_delivery.csv** (824 observations, 10 features; class = DELIV_STATUS).

1. Generate training and hold-out partitions, keeping $\frac{1}{3}$ of the data for hold-out.
2. Fit a k-Nearest Neighbor model.
   - Center and scale all predictors.

Building and Evaluating a k-Nearest Neighbor Classifier

Use the dataset **package_delivery.csv** (824 observations, 10 features; class = DELIV_STATUS).

1. Generate training and hold-out partitions, keeping $\frac{1}{3}$ of the data for hold-out.
2. Fit a k-Nearest Neighbor model.
   - Center and scale all predictors.
   - Use cross-validation to choose the optimal k.

Building and Evaluating a k-Nearest Neighbor Classifier

Use the dataset **package_delivery.csv** (824 observations, 10 features; class = DELIV_STATUS).

1. Generate training and hold-out partitions, keeping $\frac{1}{3}$ of the data for hold-out.
2. Fit a k-Nearest Neighbor model.
   - Center and scale all predictors.
   - Use cross-validation to choose the optimal k.
   - Predict the hold-out set.

Building and Evaluating a k-Nearest Neighbor Classifier

Use the dataset **package_delivery.csv** (824 observations, 10 features; class = DELIV_STATUS).

1. Generate training and hold-out partitions, keeping $\frac{1}{3}$ of the data for hold-out.
2. Fit a k-Nearest Neighbor model.
   - Center and scale all predictors.
   - Use cross-validation to choose the optimal k.
   - Predict the hold-out set.
   - Produce a confusion matrix and accuracy.

Building and Evaluating a k-Nearest Neighbor Classifier

Use the dataset **package_delivery.csv** (824 observations, 10 features; class = DELIV_STATUS).

1. Generate training and hold-out partitions, keeping $\frac{1}{3}$ of the data for hold-out.
2. Fit a k-Nearest Neighbor model.
   - Center and scale all predictors.
   - Use cross-validation to choose the optimal k.
   - Predict the hold-out set.
   - Produce a confusion matrix and accuracy.
3. Decide whether the resulting classifier is acceptable and explain why.

k-Nearest Neighbor Classifier

- **Objective** – classifies with a distance-based learner (k-NN).

k-Nearest Neighbor Classifier

- **Objective** – classifies with a distance-based learner (k-NN).
- **Train/hold-out split** – keeps evaluation unbiased; $\frac{1}{3}$ hold-out follows the holdout-method guideline.

k-Nearest Neighbor Classifier

- **Objective** – classifies with a distance-based learner (k-NN).
- **Train/hold-out split** – keeps evaluation unbiased; $\frac{1}{3}$ hold-out follows the holdout-method guideline.
- **Center & scale** – standardization ensures every numeric attribute contributes equally to distance

k-Nearest Neighbor Classifier

- **Objective** – classifies with a distance-based learner (k-NN).
- **Train/hold-out split** – keeps evaluation unbiased; $\frac{1}{3}$ hold-out follows the holdout-method guideline.
- **Center & scale** – standardization ensures every numeric attribute contributes equally to distance
- **Select k** – use cross-validation to find the k that maximizes validation accuracy; small k risks noise, large k risks over-smoothing.

k-Nearest Neighbor Classifier

- **Objective** – classifies with a distance-based learner (k-NN).
- **Train/hold-out split** – keeps evaluation unbiased; $\frac{1}{3}$ hold-out follows the holdout-method guideline.
- **Center & scale** – standardization ensures every numeric attribute contributes equally to distance
- **Select k** – use cross-validation to find the k that maximizes validation accuracy; small k risks noise, large k risks over-smoothing.
- **Confusion matrix & accuracy** – summarize predictions on the hold-out set; accuracy $= \frac{\text{TP}+\text{TN}}{\text{Total}}$, but also inspect class-wise recall and precision for balance.

k-Nearest Neighbor Code - Generating partitions

```r
df <- read.csv("package_delivery.csv")

# Ensure the class column is categorical so caret treats it correctly.
df$DELIV_STATUS <- factor(df$DELIV_STATUS)

# ── 2. Train / hold-out split (⅔ : ⅓) ──────────────────────────────
# initial_split(): stratified on DELIV_STATUS, matching the hold-out
# method depicted on slide 11 of the lecture.
library(rsample)
set.seed(42) #Optional. Makes it precisely reproducible.
split  <- initial_split(df, prop = 2/3, strata = DELIV_STATUS)
train  <- training(split)
holdout <- testing(split)
```

k-Nearest Neighbor Code - Fit, Center, Scale, Best K

```r
# caret::train(): mirrors lines ~410-420 of Module 2.R
library(caret)
ctrl <- trainControl(method = "cv",    # 10-fold CV
                     number = 10,
                     classProbs = FALSE,
                     summaryFunction = defaultSummary)

knn_mod <- train(DELIV_STATUS ~ ., data = train,
                 method      = "knn",
                 trControl   = ctrl,
                 preProcess  = c("center", "scale"),  # mandated by lecture
                 tuneLength  = 30)                    # search 30 odd k's

# Best k chosen by caret
best_k <- knn_mod$bestTune$k
```

k-Nearest Neighbor Code - Predict, Confusion Matrix, Accuracy

```r
pred <- predict(knn_mod, newdata = holdout, type = "raw")

# confusionMatrix(): same metric block used in Module 2.R
cm <- confusionMatrix(data = pred,
                      reference = holdout$DELIV_STATUS,
                      positive  = "OnTime")   # pick the “positive” label as needed

list(
  best_k          = best_k,
  confusion_table = cm$table,
  accuracy        = cm$overall["Accuracy"]
)
```

k-Nearest Neighbor Results

```
$best_k

[1] 31


$confusion_table

           Reference

Prediction  LateMajor LateMinor OnTime

  LateMajor         0         0      0

  LateMinor        10        88     12

  OnTime            0        47    119


$accuracy

Accuracy

    0.75
```

Predicting Rarer But Significant Class Values

- Sometimes in real data, a particular value appears rarely, but is still significant.

Predicting Rarer But Significant Class Values

- Sometimes in real data, a particular value appears rarely, but is still significant.
- For example, in the package dataset, very late packages (LateMajor) appear only 30 times, but its negative impact on the courier is significant.

Predicting Rarer But Significant Class Values

- Sometimes in real data, a particular value appears rarely, but is still significant.
- For example, in the package dataset, very late packages (LateMajor) appear only 30 times, but its negative impact on the courier is significant.
- One option for predicting these is binary logistic regression with over sampling.

Predicting Rarer But Significant Class Values

- Sometimes in real data, a particular value appears rarely, but is still significant.
- For example, in the package dataset, very late packages (LateMajor) appear only 30 times, but its negative impact on the courier is significant.
- One option for predicting these is binary logistic regression with over sampling.
- The rare value becomes the "Yes" or "1".

Binary Logistic Regression

- Predicts a yes/no outcome, giving a probability between 0 and 1.

Binary Logistic Regression

- Predicts a yes/no outcome, giving a probability between 0 and 1.
- If the probability passes the cutoff, we label the row “yes”.

Binary Logistic Regression

- Predicts a yes/no outcome, giving a probability between 0 and 1.
- If the probability passes the cutoff, we label the row "yes".
- Under-sampling randomly drops many "no" rows until yes/no are roughly even.

Binary Logistic Regression

- Predicts a yes/no outcome, giving a probability between 0 and 1.
- If the probability passes the cutoff, we label the row “yes”.
- Under-sampling randomly drops many “no” rows until yes/no are roughly even.
- Over-sampling copies or synthetically creates “yes” rows until row counts are even.

Applying Logistic Regression/Over-Sampling to Package Delivery

We want to answer this question: Given it's late, can we tell if it is majorly late?

1. Remove all OnTime values.

Applying Logistic Regression/Over-Sampling to Package Delivery

We want to answer this question: Given it's late, can we tell if it is majorly late?

1. Remove all OnTime values.
2. Split the dataset into training and holdout.

Applying Logistic Regression/Over-Sampling to Package Delivery

We want to answer this question: Given it's late, can we tell if it is majorly late?

1. Remove all OnTime values.
2. Split the dataset into training and holdout.
3. Apply over-sampling for the LateMajor value.

Applying Logistic Regression/Over-Sampling to Package Delivery

We want to answer this question: Given it's late, can we tell if it is majorly late?

1. Remove all OnTime values.
2. Split the dataset into training and holdout.
3. Apply over-sampling for the LateMajor value.
4. Fit a logistic regression model to the updated dataset.

Applying Logistic Regression/Over-Sampling to Package Delivery

We want to answer this question: Given it's late, can we tell if it is majorly late?

1. Remove all OnTime values.
2. Split the dataset into training and holdout.
3. Apply over-sampling for the LateMajor value.
4. Fit a logistic regression model to the updated dataset.
5. Generate a confusion matrix and compute the accuracy and the F-score to make predictions.

Majorly Late? Steps 1 - 3 Code

```r
# 1. Read the package-delivery data and drop the OnTime rows
df <- read.csv("package_delivery.csv")
df <- subset(df, DELIV_STATUS != "OnTime")          # keep only LateMinor / LateMajor
df$DELIV_STATUS <- factor(df$DELIV_STATUS)          # be sure class is a factor

# 2. Stratified train / hold-out split (²/₃ train, ¹/₃ test)
library(rsample)
set.seed(31)
split  <- initial_split(df, prop = 0.67, strata = DELIV_STATUS)
train  <- training(split)
hold   <- testing(split)

# 3. Over-sample the minority class (LateMajor) in the training set
library(ROSE)                                       # supplies ovun.sample()
train_over <- ovun.sample(DELIV_STATUS ~ .,
                          data   = train,
                          method = "over",          # simple random over-sampling
                          seed   = 33,
                          p      = 0.5)$data        # make the two classes ≈50 / 50
table(train$DELIV_STATUS)       # before
table(train_over$DELIV_STATUS)  # after

# 4. Fit a logistic regression model on the over-sampled training data
logit_mod <- glm(DELIV_STATUS ~ ., data = train_over,
                 family = "binomial")               # LateMajor is reference level
```

Majorly Late? Steps 1 - 3 Results

```
LateMajor LateMinor
       20       268
```

```
LateMinor LateMajor
      268       273
```

Majorly Late? Steps 4 - 5 Code Part 1

```r
# 4. Fit a logistic regression model on the over-sampled training data
logit_mod <- glm(DELIV_STATUS ~ ., data = train_over,
                 family = "binomial")               # LateMajor is reference level

# 5. Predict the hold-out set and evaluate
prob <- predict(logit_mod, newdata = hold, type = "response")
pred <- factor(ifelse(prob >= 0.5, "LateMajor", "LateMinor"),
               levels = levels(hold$DELIV_STATUS))

library(caret)
cm <- confusionMatrix(data = pred,
                      reference = hold$DELIV_STATUS,
                      positive  = "LateMajor")      # treat LateMajor as “yes”
acc <- cm$overall["Accuracy"]
```

Majorly Late? Steps 4 - 5 Results Part 1

```
Confusion Matrix and Statistics

           Reference
Prediction  LateMajor LateMinor
  LateMajor         8        26
  LateMinor         2       106

               Accuracy : 0.8028
                 95% CI : (0.7278, 0.8648)
    No Information Rate : 0.9296
    P-Value [Acc > NIR] : 1

                  Kappa : 0.2859

 Mcnemar's Test P-Value : 1.383e-05

            Sensitivity : 0.80000
            Specificity : 0.80303
         Pos Pred Value : 0.23529
         Neg Pred Value : 0.98148
             Prevalence : 0.07042
         Detection Rate : 0.05634
   Detection Prevalence : 0.23944
      Balanced Accuracy : 0.80152

       'Positive' Class : LateMajor
```

Majorly Late? Steps 4 - 5 Code Part 2

```r
# F-score helper (mirrors Module 2 script)
calc_measures <- function(cm_tbl) {
  tp <- cm_tbl["LateMajor", "LateMajor"]
  fp <- cm_tbl["LateMajor", "LateMinor"]
  fn <- cm_tbl["LateMinor", "LateMajor"]
  precision <- tp / (tp + fp)
  recall    <- tp / (tp + fn)
  f1        <- 2 * precision * recall / (precision + recall)
  c(precision = precision, recall = recall, F1 = f1)
}
scores <- calc_measures(cm$table)

# Print a tidy summary
list(accuracy = acc,
     precision = scores["precision"],
     recall    = scores["recall"],
     F1        = scores["F1"])
```

Majorly Late? Steps 4 - 5 Results Part 2

```
$accuracy
 Accuracy
0.8028169

$precision
precision
0.2352941

$recall
recall
   0.8

$F1
       F1
0.3636364
```

Model Metrics (Augmented Data)

- **Accuracy** **0.80** – overall correct classifications

Model Metrics (Augmented Data)

- **Accuracy** **0.80** – overall correct classifications
- **Recall** **0.80** – captures most *LateMajor* cases

Model Metrics (Augmented Data)

- **Accuracy** **0.80** – overall correct classifications
- **Recall** **0.80** – captures most *LateMajor* cases
- **Precision** **0.24** – many false alarms on *LateMajor*

Model Metrics (Augmented Data)

- **Accuracy 0.80** – overall correct classifications
- **Recall 0.80** – captures most *LateMajor* cases
- **Precision 0.24** – many false alarms on *LateMajor*
- **F1 0.36** – balance between precision & recall

Model Metrics (Augmented Data)

- **Accuracy 0.80** – overall correct classifications
- **Recall 0.80** – captures most *LateMajor* cases
- **Precision 0.24** – many false alarms on *LateMajor*
- **F1 0.36** – balance between precision & recall
- Lower accuracy is expected when you start catching the rare class.