Lecture 4

Projects: 1) Identifying Penguin Species and 2) Optical Character Recognition Academic Year 1447 Term 1 Dr. Jomana Bashatah Slides adapted from Casten Lange

Before we Begin Let Us Do A Thought Experiment

I want to find somebody to spend a Saturday afternoon with and I am looking for somebody most similar to me (nearest neighbor) in terms of:

- Gender (coded as 0 for female, and 1 for male)
- Age (coded in years)
- Outdoor sports interest (coded from 0 (no interest) to 10 (enthusiast))

(all categories matter the same to me)

Let us do the Calculation for a Similarity Score

(average absolute differences)

Sake of argument: I am male (**1**), **50** years, outdoor sports score **7**:

- first candidate a male student (21 years old) (score = 5)
- second candidate an athletic outdoor (score=**9**) women (**0**) **51** years old
- third candidate an athletic outdoor (score=**9**), man (**1**)**53** years

Average Absolute Difference

$$\text{Similarity Score} = \frac{\sum|x_i - y_i|}{n}$$

Where lower scores indicate higher similarity.

Absolute Differences for Candidate 1:

- Gender: |1 - 1| = **0**
- Age: |50 - 21| = **29**
- Sports: |7 - 5| = **2**

Average Absolute Difference:

$$\frac{0 + 29 + 2}{3} = \frac{31}{3} = \boxed{10.33}$$

Absolute Differences for Candidate 2:

- Gender: |1 - 0| = **1**
- Age: |50 - 51| = **1**
- Sports: |7 - 9| = **2**

Average Absolute Difference:

$$\frac{1 + 1 + 2}{3} = \frac{4}{3} = \boxed{1.33}$$

Normalized Calculations (0-10 Scale)

$$\text{Normalized Diff} = \frac{|x_i - y_i|}{\text{Range}_i}$$

Attribute Ranges: Gender: 0-1 (Range = 1), Age: 21-53 (Range = 32), Sports: 0-10 (Range = 10)

Normalized Differences for Candidate 1:

- Gender: $\frac{|1-1|}{1} = \frac{0}{1} = \boxed{0.00}$
- Age: $\frac{|50-21|}{32} = \frac{29}{32} = \boxed{0.91}$
- Sports: $\frac{|7-5|}{10} = \frac{2}{10} = \boxed{0.20}$

Average Normalized Difference:

$$\frac{0.00 + 0.91 + 0.20}{3} = \boxed{0.37}$$

Normalized Differences for Candidate 2:

- Gender: $\frac{|1-0|}{1} = \frac{1}{1} = \boxed{1.00}$
- Age: $\frac{|50-51|}{32} = \frac{1}{32} = \boxed{0.03}$
- Sports: $\frac{|7-9|}{10} = \frac{2}{10} = \boxed{0.20}$

Average Normalized Difference:

$$\frac{1.00 + 0.03 + 0.20}{3} = \boxed{0.41}$$

Overview

In this session you will learn:

1. What is the underlying **idea of k-Nearest Neighbors**
2. How similarity can be measured with **Euclidean distance**
3. Why **scaling predictor variables** is important for some machine learning models
4. Why the **tidymodels package** makes it easy to work with machine learning models
5. How you can define a **recipe** to pre-process data with the `tidymodels` package
6. How you can define a **model-design** with the `tidymodels` package
7. How you can create a machine learning **workflow** with the `tidymodels` package
8. How **metrics** derived from a **confusion matrix** can be used to asses prediction quality
9. Why you have to be careful when interpreting *accuracy*, when you work with **unbalanced observations**
10. How a machine learning model can **process images** and how OCR (Optical Character Recognition) works

About the Penguin Dataset

We will work with the Palmer Penguins dataset containing 344 observations about different penguin species and their morphological measurements.

Our goal is to develop a k-Nearest Neighbors model that can predict the species of a penguin (Adelie, Chinstrap, or Gentoo) based on the penguin's bill dimensions, flipper length, and body mass

Raw Observations from Penguin Dataset

```
library(tidyverse)
library(janitor)
library(palmerpenguins)
DataPenguins <- penguins
print(DataPenguins[1:10,])
```

```
# A tibble: 10 × 8
   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
 1 Adelie  Torgersen           39.1          18.7               181        3750
 2 Adelie  Torgersen           39.5          17.4               186        3800
 3 Adelie  Torgersen           40.3          18                 195        3250
 4 Adelie  Torgersen           NA            NA                  NA          NA
 5 Adelie  Torgersen           36.7          19.3               193        3450
 6 Adelie  Torgersen           39.3          20.6               190        3650
 7 Adelie  Torgersen           38.9          17.8               181        3625
 8 Adelie  Torgersen           39.2          19.6               195        4675
 9 Adelie  Torgersen           34.1          18.1               193        3475
10 Adelie  Torgersen           42            20.2               190        4250
# ℹ 2 more variables: sex <fct>, year <int>
```

Observations from Penguin Dataset for Selected Variables

Bill Length,

Note we use `clean_names("upper_camel")` from the `janitor` package to change all column (variable) names to UpperCamel.

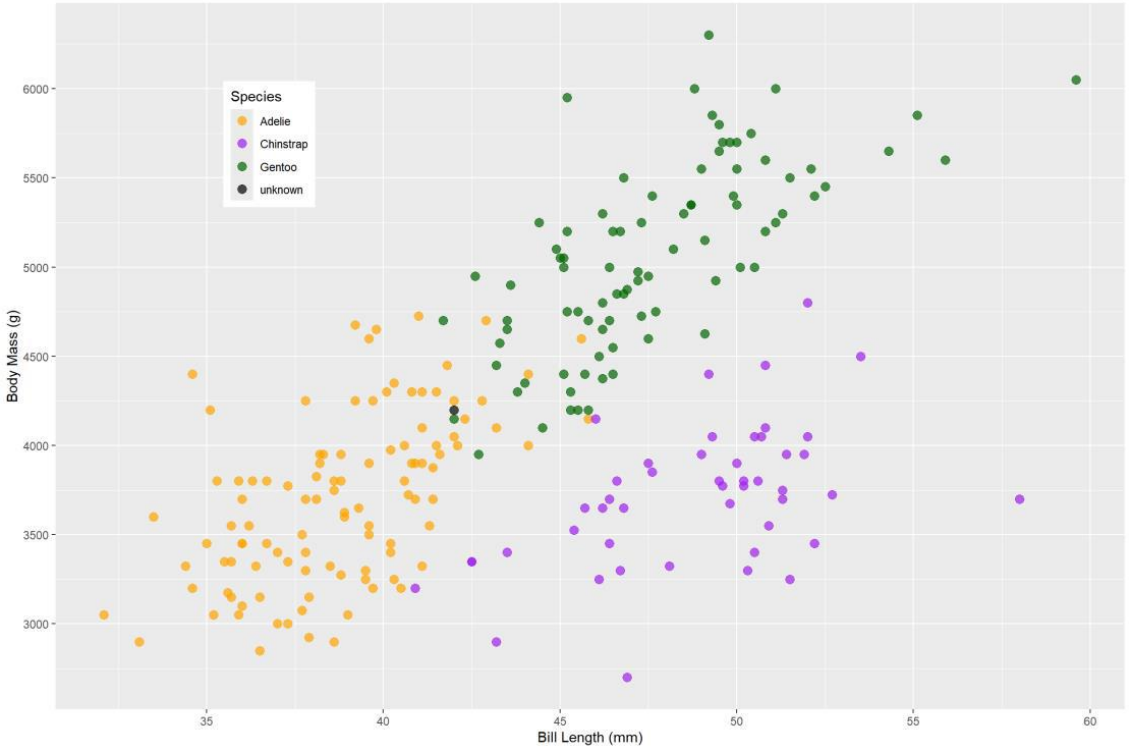
```
library(tidyverse); library(janitor)
DataPenguins <- penguins |>
  clean_names("upper_camel") |>
  select(Species, BillLengthMm, BodyMassG) |>
  drop_na() |>  # Remove missing values
  mutate(Species = as.factor(Species))
print(DataPenguins[1:10,])
```

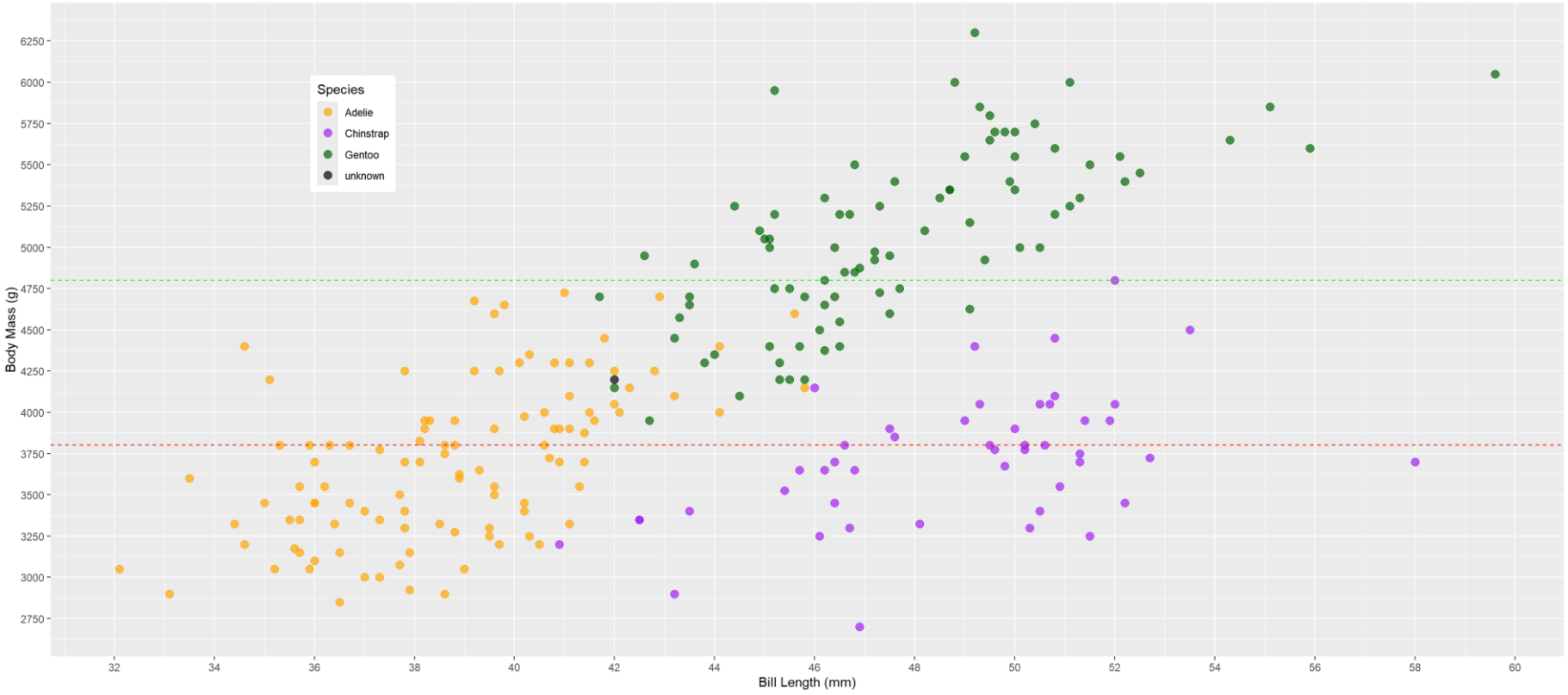
```
# A tibble: 10 × 3
   Species BillLengthMm BodyMassG
   <fct>          <dbl>     <int>
 1 Adelie          39.1      3750
 2 Adelie          39.5      3800
 3 Adelie          40.3      3250
 4 Adelie          36.7      3450
 5 Adelie          39.3      3650
 6 Adelie          38.9      3625
 7 Adelie          39.2      4675
 8 Adelie          34.1      3475
 9 Adelie          42        4250
10 Adelie          37.8      3300
```

Before Starting with k Nearest Neighbors

Let us find some eyeballing techniques that are related to various machine learning models

Eye Balling Techniques to Identify Penguin Species




Species
Adelie
Chinstrap
Gentoo
unknown
Body Mass (g)
6250
6000
5750
5500
5250
5000
4750
4500
4250
4000
3750
3500
3250
3000
2750
Bill Length (mm)
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60

Confusion Matrix

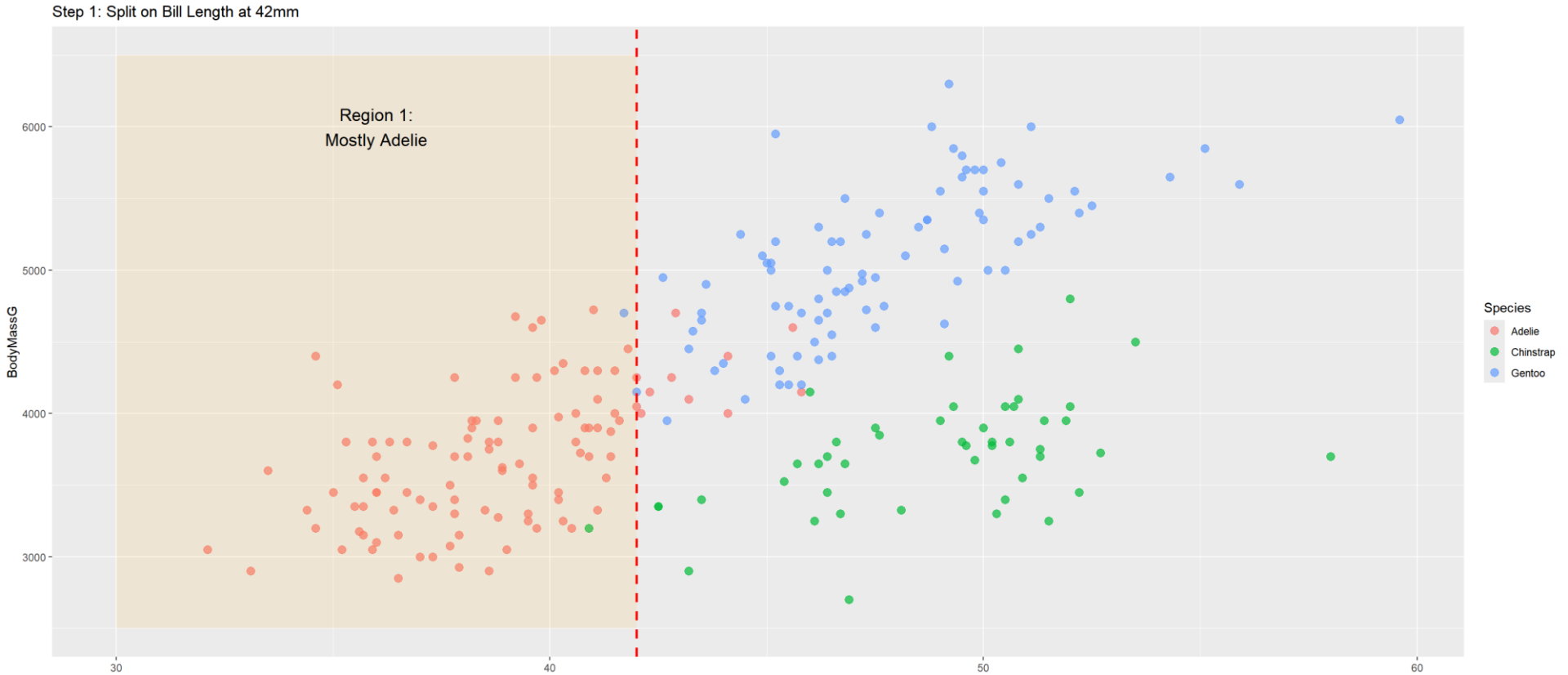
	Truth		
Prediction	Adelie	Chinstrap	Gentoo
Adelie	64	41	0
Chinstrap	31	16	0
Gentoo	0	30	56

Accuracy:

```
Overall Accuracy: 57.1 %
```

Can we improve the accuracy?

Eyeballing Techniques to Identify Penguin Species




Decision Tree-like Subspaces for Penguin Classification
Combining Vertical and Horizontal Boundaries
Species
Adelie
Chinstrap
Gentoo
Gentoo
Chinstrap
Adelie
Body Mass (g)
6000
5000
4000
3000
30
40
50
60
Bill Length (mm)

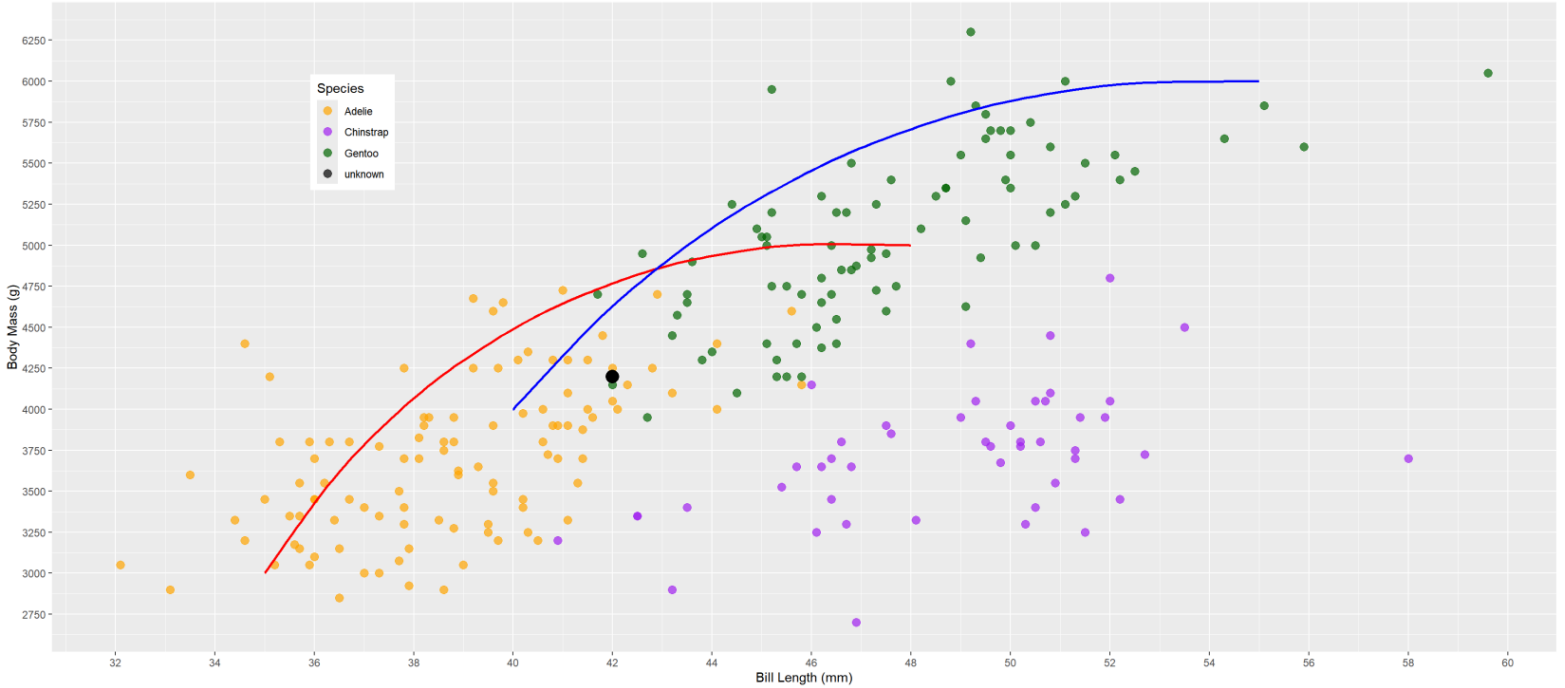
Confusion Matrix

	Predicted		
Actual	Adelie	Chinstrap	Gentoo
Adelie	96	6	3
Chinstrap	1	42	4
Gentoo	2	5	79

Tree-like Boundaries Accuracy: 91.2 %

Eyeballing Techniques to Identify Penguin Species

Using a non-linear Decision Boundary Like a Neural Network



Confusion Matrix

	Truth		
Prediction	Adelie	Chinstrap	Gentoo
Adelie	13	68	24
Chinstrap	32	14	1
Gentoo	0	5	81

Non-linear Boundary Accuracy: 45.4 %

So, how does k Nearest Neighbors Work?

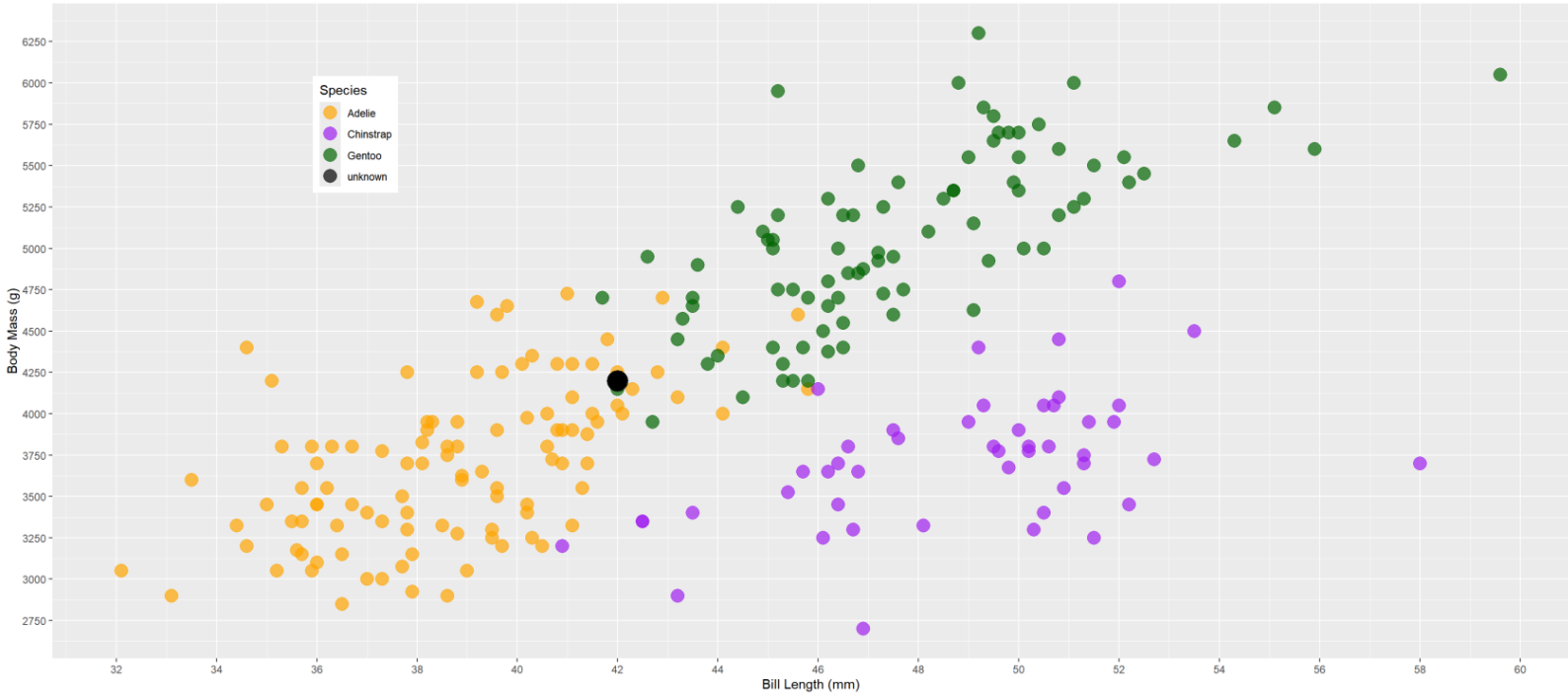
- Predicts the class (species) for a new observation (penguin with unkown Species) by finding the observation closest to it - the nearest neighbor

- If k-Nearest Neighbors considers more than one neighboring point (k > 1), e.g., the four nearest neighbors (k = 4), the class of the majority of these neighbor points is the predicted class. In case of a tie, the prediction is chosen randomly

- the hyper-parameter k determines the number of neighbors to be considered.
  - k is called a hyper-parameter because it has to be chosen at the design stage of the model.
  - In contrast to parameters that are determined based on data

k Nearest Neighbors k=1

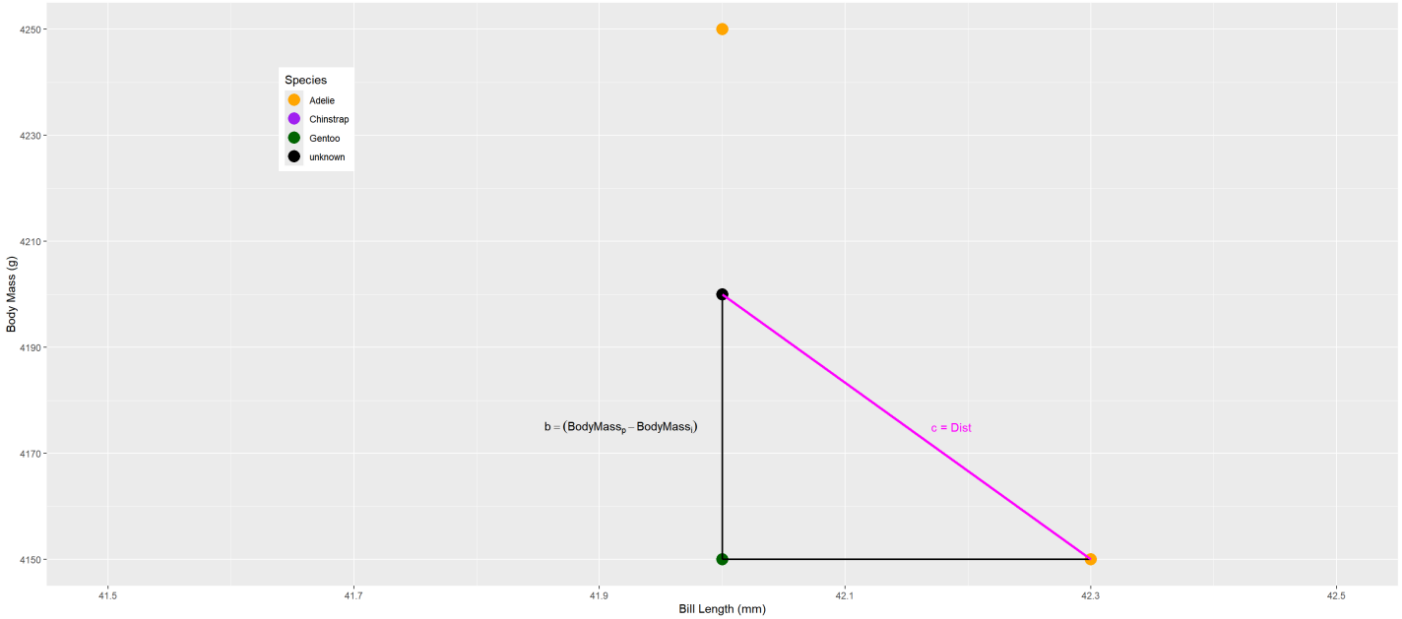
- Most basic K-Nearest Neighbors model (k=1)



k Nearest Neighbors k=1

```
[1] "Nearest neighbor at: 42.3 4150"
```

Predicting Penguin Species with k-Nearest Neighbors (k=1)



How to calculate Euclidean Distance for Two Variables

Assume our observations have **two predictor variables** x and y. We compare the unknown point p to one of the points from the training data (e,g., point i):

$$Dist_i = \sqrt{(x_p - x_i)^2 + (y_p - y_i)^2}$$

»

How to calculate Euclidean Distance for Three Variables

Assume our observations have **three predictor variables** x, y, and z. We compare the unknown point p to one of the points from the training data (e,g., point i):

$$Dist_i = \sqrt{\left(x_p - x_i\right)^2 + \left(y_p - y_i\right)^2 + \left(z_p - z_i\right)^2}$$

»

How to calculate Euclidean Distance for N Variables

Assume our observations have N predictor variables v_j with $j = 1 \ldots N$. We compare the unknown point p to one of the points from the training data (e,g., point i):

$$Dist_i = \sqrt{\sum_{j=1}^{N} \left(v_{p,j} - v_{i,j}\right)^2}$$

»

Process of Prediction using K-Nearest Neighbors

1. Take the first observation from the testing dataset and calculate the distance from this record to all observations in the training dataset.

2. Find the observation that has the smallest distance to the testing observation.

3. The predicted class (e.g., species) for the observation from the testing dataset is the same as the class from its nearest neighbor.

4. For the observation from the testing dataset, we actually know the true class — although we never showed it to the model. Therefore, we can compare the true class of the testing observation with the prediction to find out if the prediction was true or false.

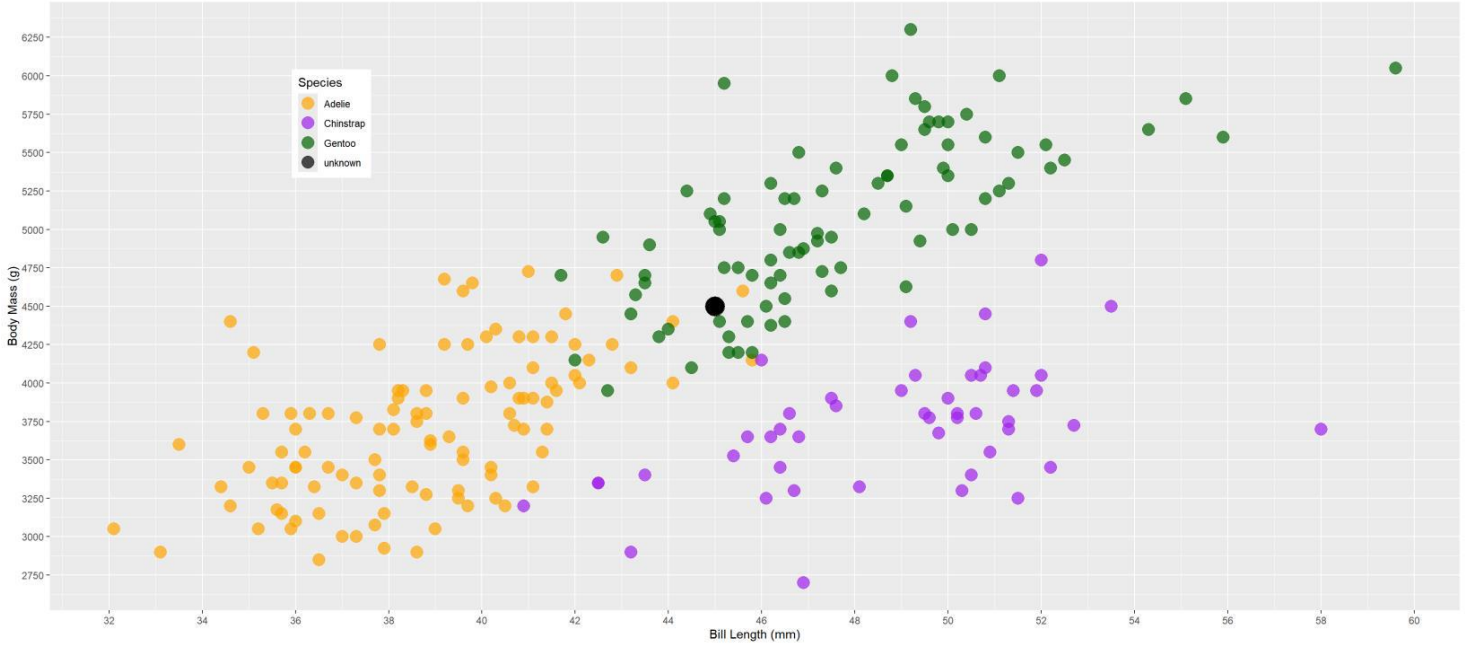
5. Depending on the prediction (red or white) and whether it was correct, we update one of the four cells in the confusion matrix.

We repeat Steps 1 – 5 for all observations from the testing dataset.

Note, when values for the outcome class (e.g., species) are unknown, Steps 4 and 5 are omitted.

k Nearest Neighbors k=4 (for a different unknown species)

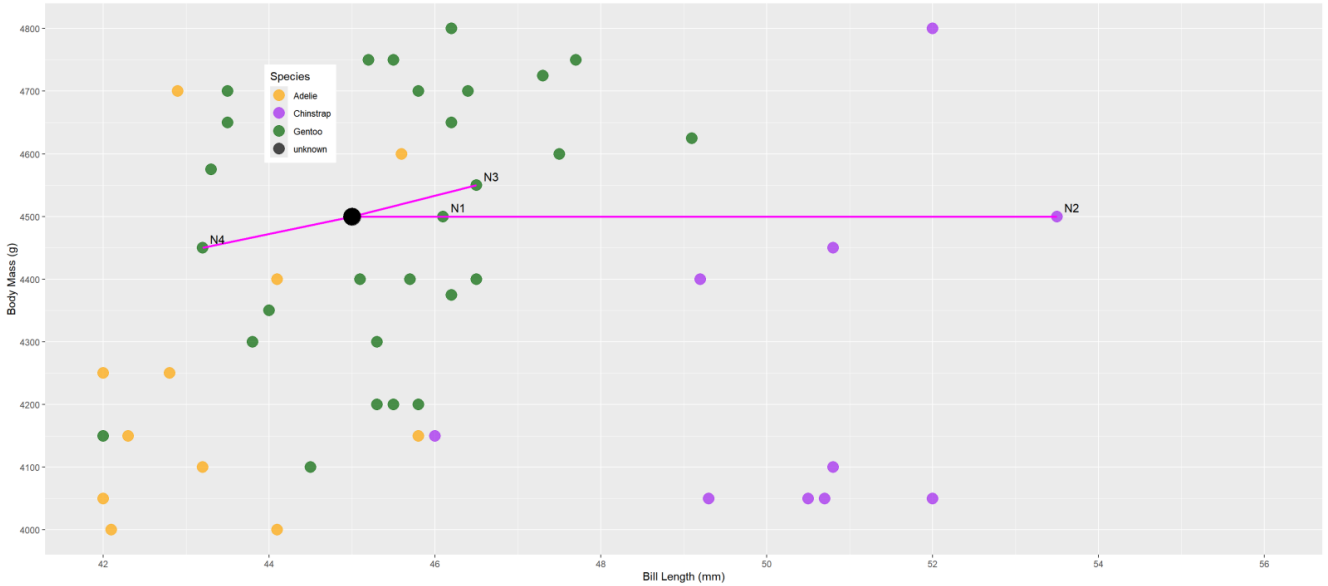
Bill Length and Body Mass Related to Penguin Species



k Nearest Neighbors k=4 (for a different unknown species)

4 nearest neighbors vote on species classification

Predicting Penguin Species with k-Nearest Neighbors (k=4)



Find the Optimal K hyperparameters

In a real-world application, you have to choose the value for the hyper-parameter
k in the model design stage. The chosen
k is then valid for all model predictions.
This raises the question: **How do we find an appropriate value for k ?**

The answer is: We use a systematic trial-and-error process called “tuning”.

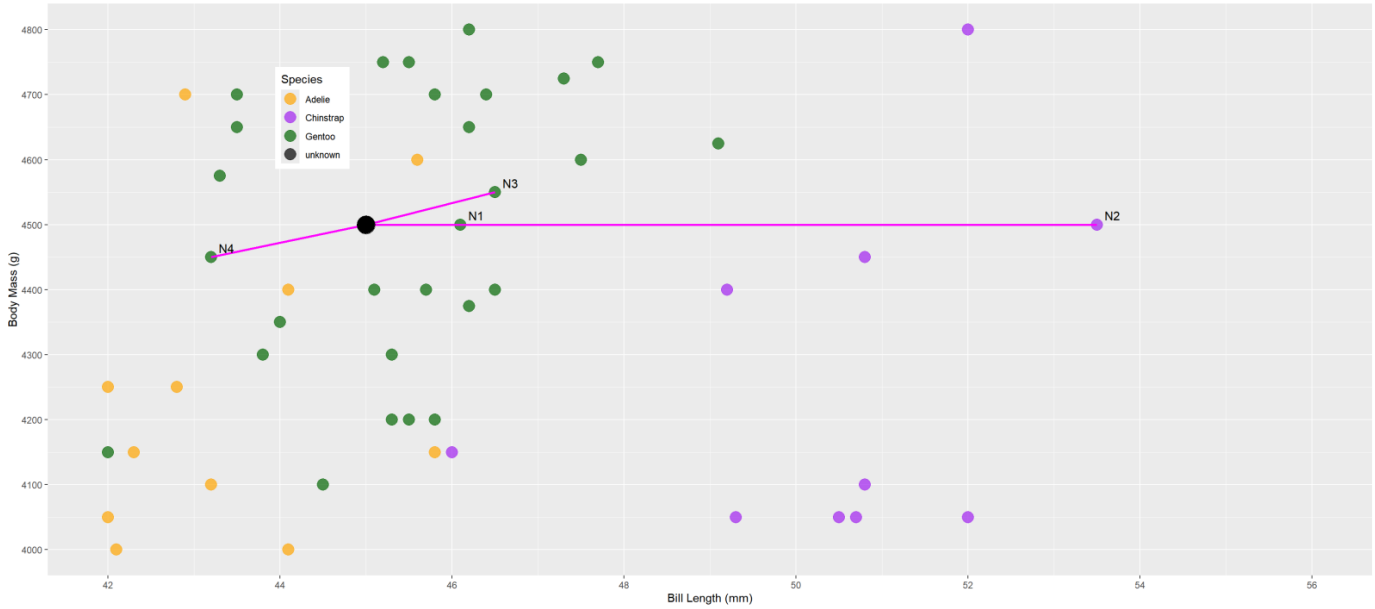
Tuning to be covered in later chapters.

- Right now, you might be tempted to run the model for different values of k and then use the testing dataset to see which k delivers the best prediction performance.
  - This is not an appropriate way to optimize hyper-parameters. Using the testing dataset to optimize hyper-parameters can lead to overfitting
- Overfitting occurs when a prediction model performs well on the training data, but when it is used for preditions based on new data that the model has “never seen before”, it performs poorly.
- In general, a k that is too low is prone to be influenced by isolated outliers, although the surrounding neighborhood would suggest otherwise.

- On the other hand, a k that is too high would consider a neighborhood so large that it does not represent the neighborhood surrounding the prediction point anymore

k Nearest Neighbors k=4 (for a different unknown species)

Watch the scale: mm vs. g. Need better scaling!



The Visual vs. The Reality

What We See:
N2 (purple point) looks very far away horizontally.
N4 (green point) looks much closer overall.
Visually, N4 should be the closer neighbor.

What The Math Says:

```
# Distance to N2 (approx at 53, 4500)
bill_diff_N2 <- 53 - 45     # = 8 mm
mass_diff_N2 <- 4500 - 4500  # = 0 g
sqrt(bill_diff_N2^2 + mass_diff_N2^2)  # ≈ 8
[1] 8
# Distance to N4 (approx at 43, 4450)
bill_diff_N4 <- 43 - 45     # = -2 mm
mass_diff_N4 <- 4450 - 4500  # = -50 g
sqrt(bill_diff_N4^2 + mass_diff_N4^2)  # ≈ 50
[1] 50.03998
```

The Scaling Problem

Different Measurement Scales:

Variable	Units	Typical Range	Example Difference
Bill Length	millimeters (mm)	30-60	0.1 - 2 mm
Body Mass	grams (g)	2500-6500	20 - 500 g

The Problem:

- Body Mass values are naturally **100-1000× larger** than Bill Length values
- When we square these differences for Euclidean distance, the gap becomes **10,000-1,000,000× larger**
- Body Mass **completely dominates** the distance calculation

This is unfair! Both measurements should contribute appropriately to finding the nearest neighbor.

We Need to Scale Our Variables

Goal: Transform both variables to comparable ranges so neither dominates the distance calculation.

Before Scaling:

Bill Length: 42.1 - 42.0 = 0.1 → contributes 0.01 to distance²

Body Mass: 4220 - 4200 = 20 → contributes 400 to distance²

Body Mass dominates (99.997% of total distance)

A Few Common Scaling Options

- **Same units**

  Divide or multiply to get the same units. This is often not possible (e.g., `BillLength` and `BodyMass`). Or it is not feasible (e.g. `BillLength` in mm and `BodyMass` in grams are in vastly different ranges)

- **Rescaling**

  Generates a variable y that is scaled to a range between 0 and 1 based on the original variable's value x, its minimum x_{min} and its maximum x_{max}:

$$y = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- **Z-Score Normalization**

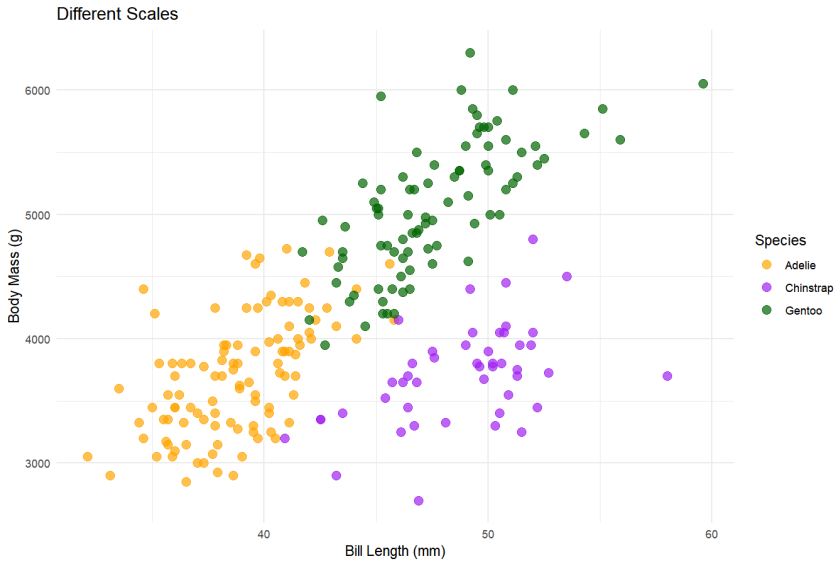
  Z-score normalization uses the mean ($\overline{x}$) and the standard deviation (s) of a variable to scale the variable x to the variable z:

$$z = \frac{x - \overline{x}}{s}$$

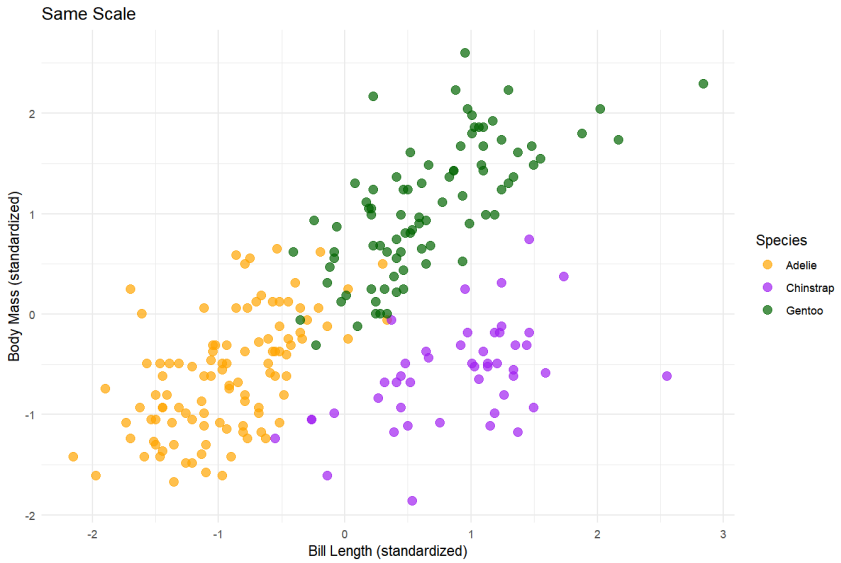
»

Comparison: Before and After Scaling

Raw Values (Problem)



Standardized Values (Solution)



Time to Run k-Nearest Neighbors

Loading Data and Selecting Variables

```
library(tidyverse); library(janitor); library(palmerpenguins)
DataPenguins <- penguins %>%
          clean_names("upper_camel") %>%
          select(Species, BillLengthMm, BodyMassG) %>%
          drop_na() %>%
          mutate(Species = as.factor(Species))
print(DataPenguins[1:10,])  # Shows first 10 rows
```

```
# A tibble: 10 × 3
   Species BillLengthMm BodyMassG
   <fct>          <dbl>     <int>
 1 Adelie          39.1      3750
 2 Adelie          39.5      3800
 3 Adelie          40.3      3250
 4 Adelie          36.7      3450
 5 Adelie          39.3      3650
 6 Adelie          38.9      3625
 7 Adelie          39.2      4675
 8 Adelie          34.1      3475
 9 Adelie          42        4250
10 Adelie          37.8      3300
```

Time to Run k-Nearest Neighbors

The `tidymodels` package provides a **standardized workflow** with standardized commands for the following tasks:

Data splitting (training and testing)

Pre-processing data with `recipes`

Creating machine learning model-design with only three standardized commands

Tuning hyper-parameters

assessing prediciton quality using a set of predefined metrics

Generate Training and Testing Data (Splitting):

```
library(tidymodels)
set.seed(876)
Split7030 =
initial_split(DataPenguins, prop=0.7,
strata = Species)
DataTrain = training(Split7030)
DataTest = testing(Split7030)
head(DataTrain)
```

Species	BillLengthMm	BodyMassG
Adelie	40.3	3250
Adelie	36.7	3450
Adelie	39.3	3650
Adelie	38.9	3625
Adelie	39.2	4675
Adelie	42.0	4250

Time to Run k-Nearest Neighbors

[Click here to find a reference list for various Step_ commands]

- Recipes simplify data pre-processing

- You can compare a `tidymodels` recipe to a recipe in a cookbook.
  - First, the ingredients are listed, and then you find the steps to use these ingredients to cook the meal.

Recipe: Prepare Data for Analysis:

```
DataTrain <- DataTrain %>% select(Species, BillLengthMm, BodyMassG)
RecipePenguins = recipe(Species ~ BillLengthMm + BodyMassG, data = DataTrain)
|>
  step_naomit() |>
  step_normalize(all_predictors())
```

The recipe() command is followed by instructions on how to process the data step by step. Each step starts with step_, indicating that the instruction (command) is part of a recipe.

Or:

```
RecipePenguins = recipe(Species~., data = DataTrain) |>
  step_naomit() |>
  step_normalize(all_predictors())
print(RecipePenguins)
```

When to Use Recipes and When to Use Select() and Mutate()

1. Generally, using a recipe is advisable because we can reuse a recipe on other data frames.
2. It is good practice to use select() before a recipe to reduce the columns of an original data frame to only those columns (variables) that are required for the analysis.
- This allows us to use the .-notation in the formula argument of the recipe() command.
3. When transforming outcome variables, it is advised to always do this outside of a recipe. For example, in the R code above, we used mutate() to convert the outcome variable Species from a `character` data type to a `factor` data type outside the recipe.

- If a recipe is later used on another dataset for prediction, this dataset might not contain a column for the outcome variable.

Time to Run k-Nearest Neighbors

Click here to find a reference list for various ML algorithm commands

a model-design determines which machine learning model from which R package should be used.

To define a model design within the `tidymodels` environment, only three commands (connected with |>) are required.

Creating a Model Design:

```
ModelDesignKNN = nearest_neighbor(neighbors = 4, weight_func = "rectangular") |>
  set_engine("kknn") %>% #provides the name of the package that performs the ML algorithm
  set_mode("classification") #indicates if we perform classification or regression
print(ModelDesignKNN)
```

```
K-Nearest Neighbor Model Specification (classification)

Main Arguments:
  neighbors = 4
  weight_func = rectangular

Computational engine: kknn
```

Time to Run k-Nearest Neighbors

So far, we've defined a `recipe` and a `model-design`
We put it all together in a `workflow`
Then, the workflow is fiited (calibrated) to the training data
Putting it all together in a **fitted** `workflow`:

```
WFModelPenguins = workflow() |>
  add_recipe(RecipePenguins) |>
  add_model(ModelDesignKNN) |>
  fit(DataTrain)

print(WFModelPenguins)
```

```
══ Workflow [trained] ══════════════════════════════════════════════════════════
Preprocessor: Recipe
Model: nearest_neighbor()

── Preprocessor ────────────────────────────────────────────────────────────────
2 Recipe Steps

• step_naomit()
• step_normalize()

── Model ───────────────────────────────────────────────────────────────────────

Call:
kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(4,     data, 5), kernel = ~"rectangular")

Type of response variable: nominal
Minimal misclassification: 0.05882353
Best kernel: rectangular
Best k: 4
```

Time to Run k-Nearest Neighbors

How to use the **fitted** `workflow` to predict the penguin species for the penguins in the testing dataset:

1. Start with observation $i = 1$ from `DataTest` (the first observation).
2. Take observation i from `DataTest` and use `BillLength` and `BodyMassG` to calculate the Euclidean distance to **each** of the observations of `DataTrain`.
3. Isolate the 4 observations with the smallest Euclidean distance and use the majority of their species as a prediction for observation i from `DataTest` (in case of a tie, decide randomly).
4. Increase i by one (i.e., take the next observation from `DataTest`) and go to step 2 (until all `DataTest` observations are processed).

Time to Run k-Nearest Neighbors

Predicting with the fitted workflow using `predict()` (not exactly helpful!):

```
DataPred = predict(WFModelPenguins,
DataTest)
head(DataPred)
```

.pred_class
Adelie
Adelie
Adelie
Adelie
Adelie
Adelie

Note: we cannot see if the predictions are correct because we cannot easily compare

Time to Run k-Nearest Neighbors

Predicting with the fitted workflow using `augment()` which *augments* `DataTest` with the predictions:

```
DataPredWithTestData=augment(WFModelPenguins, DataTest)
head(DataPredWithTestData)
```

.pred_class	.pred_Adelie	.pred_Chinstrap	.pred_Gentoo	Species	BillLengthMm	BodyMassG
Adelie	1.00	0.00	0	Adelie	39.1	3750
Adelie	1.00	0.00	0	Adelie	39.5	3800
Adelie	1.00	0.00	0	Adelie	34.1	3475
Adelie	0.75	0.25	0	Adelie	41.1	3200
Adelie	1.00	0.00	0	Adelie	36.6	3700
Adelie	1.00	0.00	0	Adelie	38.7	3450

Having a Data Frame with `truth` and `estimate` we can calculate **performance metrics**

The `tidymodels` package provides several commands to calculate metrics that reflect predictive performance.

Most of these commands compare the `estimate` with the `truth` and then calculate the related metrics.

We can use the `conmat()` command to create the confusion matrix

Confusion Matrix:

```
ConfMatrixPenguins=conf_mat(DataPredWithTestData, truth = Species,
estimate = .pred_class)
print(ConfMatrixPenguins)
```

```
            Truth
Prediction   Adelie Chinstrap Gentoo
  Adelie         42         2      1
  Chinstrap       1        19      0
  Gentoo          3         0     36
```

Reading the Confusion Matrix

- **Key Insight:** Calculate TP, FP, FN, TN *for each class separately*
- Think **“One-vs-Rest”** for each class

Understanding the Metrics

TP: Correctly predicted as THIS class

FP: Incorrectly predicted as THIS class

FN: Actually WAS this class, but we predicted it as something else

TN: Correctly identified as NOT this class

Understanding the Metrics

To make sure the accuracy rate is not misleading, we look at `accuracy()`, `sensitivity()`,`precision()` and `specificity()` for the penguin data.

- Sensitivity: the rate of correctly predicted positives
  - "Of all the actual positives, how many did we catch?"
- Precision: the rate of correct positive predicitons
  - "Of all our positive predictions, how many were correct?"
- Specificity: the rate of correctly predicted negatives
  - "Of all the actual negatives, how many did we correctly"

Note: In our case (3 classes), sensitivity and specificity need to be calculated for each class

For each penguin species, calculate separately

Example: Adelie Penguins

“Is this penguin an Adelie?”

- TP = 42: Correctly identified as Adelie
- FP = 3: Said “Adelie” but wrong (2+1)
- FN = 4: Missed Adelie penguins (1+3)
- TN = 55: Correctly NOT Adelie (19+0+0+36)

Precision = TP/(TP+FP) = 42/(42+3) = 42/45 = 93.3%

Recall (Sensitivity) = TP/(TP+FN) = 42/(42+4) = 42/46 = 91.3%

Specificity = TN/(TN+FP) = 55/(55+3) = 55/58 = 94.8%

Example: Chinstrap Penguins

"Is this penguin a Chinstrap?"

- TP = 19: Correctly identified as Chinstrap
- FP = 1: Said "Chinstrap" but wrong (1+0)
- FN = 2: Missed Chinstrap penguins (2+0)
- TN = 82: Correctly NOT Chinstrap (42+1+3+36)

Precision = TP/(TP+FP) = 19/(19+1) = 19/20 = 95.0%

Recall (Sensitivity) = TP/(TP+FN) = 19/(19+2) = 19/21 = 90.5%

Specificity = TN/(TN+FP) = 82/(82+1) = 82/83 = 98.8%

Example: Gentoo Penguins

“Is this penguin a Gentoo?”

- TP = 36: Correctly identified as Gentoo
- FP = 3: Said “Gentoo” but wrong (3+0)
- FN = 1: Missed Gentoo penguins (1+0)
- TN = 64: Correctly NOT Gentoo (42+2+1+19)

Precision = TP/(TP+FP) = 36/(36+3) = 36/39 = 92.3%

Recall (Sensitivity) = TP/(TP+FN) = 36/(36+1) = 36/37 = 97.3%

Specificity = TN/(TN+FP) = 64/(64+3) = 64/67 = 95.5%

Let’s Look at Metrics in R

.metric	.estimator	.estimate
accuracy	multiclass	0.9326923

.metric	.estimator	.estimate
sensitivity	macro	0.9302595

```
...
sensitivity(DataPredWithTestData,
truth = Species, estimate =
.pred_class)
```

.metric	.estimator	.estimate
specificity	macro	0.9638172

```
...
specificity(DataPredWithTestData,
truth = Species, estimate =
.pred_class)
```

.metric	.estimator	.estimate
precision	macro	0.9354701

```
precision(DataPredWithTestData,
truth = Species, estimate =
.pred_class)
```

Get all the metrics at Once

```
metrics_summary <-
metric_set(sensitivity,
specificity, precision, recall)
metrics_summary(DataPredWithTestDat
a, truth = Species, estimate =
.pred_class)
```

.metric	.estimator	.estimate
sensitivity	macro	0.9302595
specificity	macro	0.9638172
precision	macro	0.9354701
recall	macro	0.9302595

When to Use Each Metric: Examples

Medical Diagnosis (Cancer Detection)
- Prioritize: **Recall (Sensitivity)**
- Don’t miss any cancer cases
- Missing a positive case (FN) is very costly

Spam Email Filter
- Prioritize: **Precision**
- Don’t mark important emails as spam
- False positives (legitimate email marked as spam) are costly

COVID-19 Screening Test
- Prioritize: **Specificity**
- Correctly identify healthy people
- False positives cause unnecessary quarantine and anxiety

Balanced Dataset (Equal class sizes)
- Use: **Accuracy**
- Simple and interpretable
- All error types have similar costs

Imbalanced Dataset (e.g., fraud detection: 99% normal, 1% fraud)
- Avoid: Accuracy (can get 99% by predicting “normal” every time!)
- Use: **Precision, Recall, and F1-Score**

The Precision-Recall Trade-off

There's often a trade-off:
- Increasing **Recall** (catch more positives) → often decreases **Precision** (more false alarms)
- Increasing **Precision** (fewer false alarms) → often decreases **Recall** (miss some positives)

F1-Score balances both:

```
F1-Score = 2 × (Precision × Recall) / (Precision + Recall)
```

- Harmonic mean of Precision and Recall

- Good when you need a single metric that balances both

Our Penguin Example: - Adelie: F1 = 2 × (0.933 × 0.913) / (0.933 + 0.913) = **0.923**
- Chinstrap: F1 = 2 × (0.905 × 0.950) / (0.905 + 0.950) = **0.927**
- Gentoo: F1 = 2 × (0.973 × 0.923) / (0.973 + 0.923) = **0.947**

PROJECT: DESIGN A MACHINE LEARNING WORKFLOW FOR OPTICAL CHARACTER RECOGNITION »

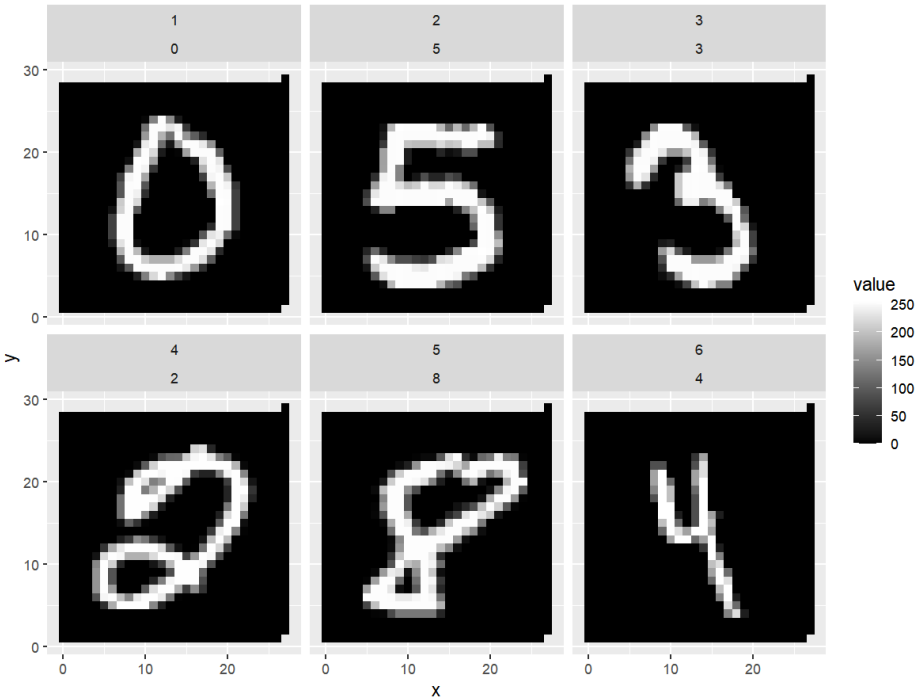
MNIST Data Set

You will develop a machine learning model based on k-Nearest Neighbors to recognize handwritten digits from images.

You will use the MNIST dataset, a standard dataset for image recognition in machine learning (60,000 images for training and 10,000 images for testing). Developed by LeCun, Cortes, and Burges (2010) based on two datasets from handwritten digits obtained from Census workers and high school students.

We will use only the first 500 images of the original MNIST dataset to speed up the *k-Nearest Neighbors* model's training time.

Visualization of the First Six Images from the MNIST Data Set



How a Image is Stored in the Mnist Dataset

The image has 28 rows and 28 columns. Each of the 784 cells (pixels) holds a value between 0 (black) and 255 (white)

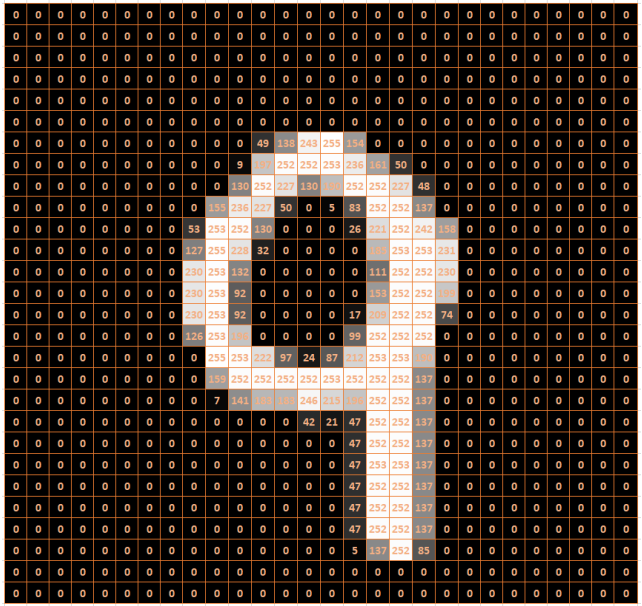


Image of a Handwritten Nine

How a Image is Stored in the Mnist Dataset

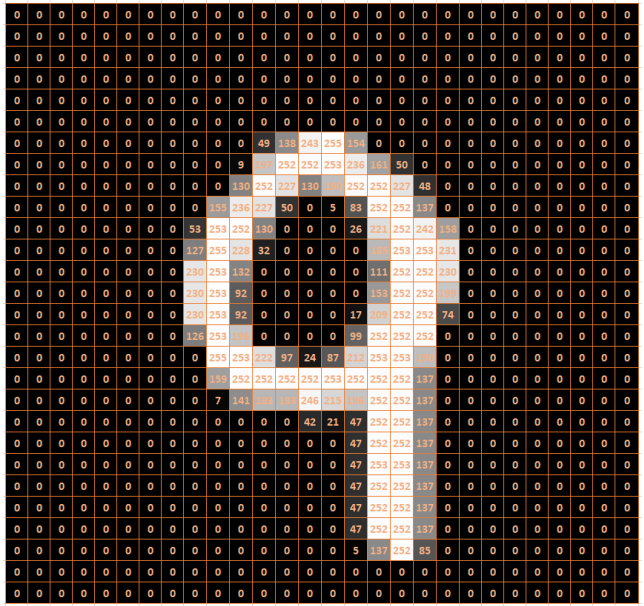


Image of a Handwritten Nine

- Pixel values for a single image are not stored in a table. Ohterwise we would end-up with a table containing tables.
- Pixel values are stored as one row for each image.
- Concatenating the 28 rows of an image into one row with 28*28=784 cells (pixels)

Go to Project in Book

Build your own OCR system.»

See: RandRStudioScriptPart1.R script300